# はじめに

皆さん，プログラミングの世界へようこそ。本講座は，コンピュータ言語の基本から，高度な技術までを学びます。プログラミングは，論理的思考と創造性の融合です。初級では，基本的な命令文の理解から始めます。中級では，コード・スニペット（短いコードのまとまり）を学習し，実際の問題に適用する方法を探究します。そして上級では，長文の問題をどのようにプログラムに落とし込むかを学びます。

AI技術の進化に伴い，プログラミングの重要性が増しています。皆さんが学ぶ知識は，将来的にAIと共存し，それを活用する能力を高めるために不可欠です。プログラミングは，単にコードを書くことではなく，問題を分析し，効率的な解決策を見つけるプロセスです。

この講座を通じて，AIの時代における新しい価値創造への一歩を踏み出しましょう。好奇心を持って学ぶことが，未知の領域を探究し，新たな発見を生み出す原動力になります。創造性，そして進化する技術との共生を学びながら，好奇心を糧にプログラミングの魅力を存分に味わってください。皆さんの成功への好奇心が，学びをより深いものにし，新たな可能性を開く鍵となるでしょう。

安藤昇

# 講座の受け方

## 受講対象

### 高1・高2

- 基礎～発展レベルのプログラミングについて理解し，自分でプログラムを作成する力を身につけたい方

### 高3・大学受験生

上記に加えて

- 共通テストなど，大学入試におけるプログラミング問題について，苦手を克服したい方，高得点を目指したい方

※情報 I の全分野について学びたい方は，教科書や通年講座の学習を進めてから，本講座を受講することをおすすめします。

## 講座の構成

本講座は，扱う題材の難易度に応じて，3つのレベルに分かれています。

- 初級編　プログラミングの基礎知識
- 中級編　いくつかの基礎知識を組み合わせるなど，やや難しいプログラミング
- 上級編　複数の視点を組み合わせるものや，構造が複雑なものなど，難しいプログラミング

※初級編は，プログラミングについて 0 から学びたい方に受講をおすすめします。ほかの講座や学校の授業で，基礎レベルのプログラミングについて十分に理解している方は，中級編からの受講をおすすめします。

## 学習の流れ

- 予習　初級編　　　　　：予習は不要<br>　　　中級編・上級編：まずは例題を解き，その後動画を見ましょう。
- 授業　動画を見て学びましょう。
- 復習　授業で学んだ内容を，確認テストで復習しましょう。

## 使用するプログラミング言語

本講座では，プログラミング言語「Python」を用いて授業を行いますが，共通テストや，汎用的なプログラミングの知識を身につけるためにもご活用いただけます。

可能であれば，ぜひ手を動かして，自分でもプログラミングをしてみてください。動画内では「Google Colaboratory」という実行環境を利用していますが，基本的にはどの環境でも同じように実行できるプログラムを扱っています。

# 目次


## 初級編

第１講　print関数
第２講　四則演算
第３講　累乗・商・余り
第４講　比較演算子
第５講　配列
第６講　乱数
第７講　len関数
第８講　range関数
第９講　for文
第10講　if文
第11講　if-else文
第12講　if-elif-else文
第13講　while文
第14講　論理演算子

## 中級編

第15講　関数
第16講　配列と乱数
第17講　２次元配列
第18講　inputとrandom.choice
第19講　引数が配列の関数
第20講　１からｎまでの和
第21講　○以上○以下の判定
第22講　２つの数字の並べ替え
第23講　合計と平均
第24講　for文の入れ子
第25講　for文とif文の入れ子
第26講　最大値と最小値
第27講　カウントダウン
第28講　while文とif文の入れ子
第29講　10進法から２進法への変換
第30講　ソート（並べ替え）
第31講　配列への要素の追加
第32講　２次元配列の応用
第33講　辞書型

## 上級編

第34講　コインの枚数
第35講　駐車料金の判定
第36講　割引のある料金
第37講　テストの集計
第38講　じゃんけん
第39講　サイコロの出目
第40講　中央値の判定
第41講　素数判定
第42講　素因数分解
第43講　倍数の判定・FizzBuzz
第44講　バブルソート・選択ソート
第45講　グラフ理論
第46講　待ち行列
第47講　パリティチェック
第48講　すごろくゲーム

初級編

# 第 1 講 print 関数

## 要点整理

### print 関数

print 関数は，( ) の中に指定した値を表示する。' ' で囲まれた文字列，数値，変数の値などを表示することができる。

```
# 文字列型
print('shibuya')

# 整数型
print(109)

# 変数に代入して出力
chimei = 'shibuya'
print(chimei)
```

実行結果

```
shibuya
109
shibuya
```

# 初級編 第 1 講 確認問題

変数 name に 'Tokyo' が代入されている。 この変数を print 関数を使って表示するプログラムとして最も適切なものを，次の①～④のうちから一つ選びなさい。

```
name = 'Tokyo'
```

① `print(Tokyo)`

② `print('Tokyo')`

③ `print(name)`

④ `print('name')`

初級編

# 第 2 講　四則演算

## 要点整理

### 四則演算

a=5，b=3 のとき

| 演算 | 演算子の使い方 | 出力 | 演算子の読み方 |
|---|---|---|---|
| 足し算 | a + b | 8 | + プラス |
| 引き算 | a - b | 2 | - マイナス |
| 掛け算 | a * b | 15 | * アスタリスク |
| 割り算 | a / b | 1.6666666666666667 | / スラッシュ |

```python
a = 5
b = 3

# 足し算
print(a + b)

# 引き算
print(a - b)

# 掛け算
print(a * b)

# 割り算
print(a / b)
```

実行結果

```
8
2
15
1.6666666666666667
```

# 初級編　第2講　確認問題

お店で１個200円のりんごを３個，１個100円のみかんを２個買い，その際50円の割引クーポンを使うとする。この支払額を表示するプログラムを作成した。
空欄に入る最も適切なものを，次の①～④のうちから一つ選びなさい。

```
# 支払額
ringo = 200
mikan = 100
waribiki = 50
print(ringo * 3 + mikan * 2 □ waribiki, '円の支払い')
```

実行結果

```
750 円の支払い
```

①　+　　②　-　　③　*　　④　/

初級編

# 第3講 累乗・商・余り

## 要点整理

### 累乗・商・余り

a=5，b=3 のとき

| 演算 | 演算子の使い方 | 出力 | 演算子の読み方 |
|---|---|---|---|
| 累乗 | a ** b | 125 | ** ダブルアスタリスク |
| 商 | a // b | 1 | // ダブルスラッシュ |
| 余り | a % b | 2 | % パーセント |

```
a = 5
b = 3

# 累乗
print(a ** b)

# 商
print(a // b)

# 余り
print(a % b)
```

実行結果

```
125
1
2
```

# 初級編　第３講　確認問題

25 を 7 で割ったときの商と余りを出力するプログラムとして最も適切なものを，次の①～④のうちから一つ選びなさい。

①

```
wararerukazu = 25
warukazu = 7
q = wararerukazu // warukazu
r = wararerukazu / warukazu
print('商', q, '余り', r)
```

②

```
wararerukazu = 25
warukazu = 7
q = wararerukazu / warukazu
r = wararerukazu % warukazu
print('商', q, '余り', r)
```

③

```
wararerukazu = 25
warukazu = 7
q = wararerukazu / warukazu
r = wararerukazu // warukazu
print('商', q, '余り', r)
```

④

```
wararerukazu = 25
warukazu = 7
q = wararerukazu // warukazu
r = wararerukazu % warukazu
print('商', q, '余り', r)
```

実行結果

```
商 3 余り 4
```

初級編

# 第 4 講 比較演算子

## 要点整理

### 比較演算子

a=5, b=3 のとき

| 演算子 | 説明 | 例 | 結果 |
|---|---|---|---|
| > | より大きい | a > b | True |
| < | より小さい | a < b | False |
| == | 等しい | a == b | False |
| != | 等しくない | a != b | True |
| >= | 以上 | a >= b | True |
| <= | 以下 | a <= b | False |

```python
# 整数型の比較
a = 5
b = 3

print(a > b)  # True
print(a < b)  # False
print(a == b) # False
print(a != b) # True
print(a >= b) # True
print(a <= b) # False
```

実行結果

```
True
False
False
True
True
False
```

# 初級編　第4講　確認問題

a＝7，b＝2のとき，a != b の結果として最も適切なものを，次の①～④のうちから一つ選びなさい。

①　`True`　　②　`False`　　③　`5`　　④　`3`

初級編

# 第 5 講 配列

## 要点整理

### 配列

配列とは，並べた同一の型のデータを扱うための変数のことである。

```
Num = [1, 2, 3, 4, 5]
print(Num)
```

実行結果

```
[1, 2, 3, 4, 5]
```

### 添字（インデックス）

配列からデータを取り出すには，添字（インデックス）を使用する。

Python では，添字は 1 ではなく 0 から始まることに注意が必要である。

```
# 例： 配列からデータを取り出す

Num = [1, 2, 3, 4, 5]

# 添字 0 の要素を取り出す
print(Num[0])

# 添字 4 の要素を取り出す
print(Num[4])
```

実行結果

```
1
5
```

# 初級編　第5講　確認問題

下記のプログラムの実行結果として最も適切なものを，
次の①～④のうちから一つ選びなさい。

```
Numbers = [1, 2, 3, 4, 5]
print(Numbers[2])
```

①　1　　②　2　　③　3　　④　4

初級編

# 第６講 乱数

## 要点整理

### 乱数

乱数とは，一定の確率分布に従って生成される数値のことで，計算やシミュレーション，データ解析などさまざまな場面で使用される。Python では，random モジュールを呼び出すことで，乱数が使用できるようになる。

```
import random

# 1 から 10 の間でランダムな整数を生成
random_int = random.randint(1, 10)
print(random_int)
```

実行結果

```
8
```

※実行するたびに結果は異なる。

# 初級編　第 6 講　確認問題

次のプログラムは，サイコロを模倣したものである。

空欄に入る最も適切なものを，次の①～④のうちから一つ選びなさい。

```
import random

dice_roll = random.□(1, 6)
print(dice_roll)
```

① randint

② randrange

③ randomint

④ randomfloat

初級編

# 第 7 講 len 関数

## 要点整理

### len 関数

len 関数は，引数に指定した配列や文字列などの長さ（配列の場合は要素数）を取得することができる。

```
# 文字列の長さを取得
print(len('Hello, world!'))
```

実行結果

```
13
```

```
# 配列の要素数を取得
Numbers = [1, 2, 3, 4, 5]
print(len(Numbers))
```

実行結果

```
5
```

# 初級編　第7講　確認問題

配列 Animals = ['cat', 'dog', 'fish', 'bird'] が与えられたとき，その要素数を求めるプログラムとして最も適切なものを，次の①～④のうちから一つ選びなさい。

① `Animals.length()`

② `len(Animals)`

③ `length(Animals)`

④ `sizeof(Animals)`

初級編

# 第 8 講 range 関数

## 要点整理

### range 関数

一定の規則に基づいた整数を生成する関数。

| 引数の数 | range 関数 | 説明 | 例 | 結果 |
|---|---|---|---|---|
| 引数が 1 個 | range(end) | 0 から end-1 までの整数 | range(5) | [0, 1, 2, 3, 4] |
| 引数が 2 個 | range(start, end) | start から end-1 までの整数 | range(3, 8) | [3, 4, 5, 6, 7] |
| 引数が 3 個 | range(start, end, step) | start から end-1 まで step 間隔の整数 | range(2, 10, 2) | [2, 4, 6, 8] |

```
# list 関数で表示
list(range(5))
```

実行結果

```
[0, 1, 2, 3, 4]
```

```
list(range(3, 8))
```

実行結果

```
[3, 4, 5, 6, 7]
```

```
list(range(2, 10, 2))
```

実行結果

```
[2, 4, 6, 8]
```

# 初級編　第８講　確認問題

[1, 4, 7, 10, 13, 16, 19] のような，一定の規則に基づいて並んだ整数を出力するプログラムとして最も適切なものを，次の①～④のうちから一つ選びなさい。

① `list(range(1, 19, 2))`

② `list(range(1, 19, 4))`

③ `list(range(1, 20, 2))`

④ `list(range(1, 20, 3))`

初級編

# 第9講 for 文

## 要点整理

### for 文

for 文に続くインデントされたブロック内の処理を繰り返し実行する制御構文。

```
for i in [0, 1, 2, 3, 4]:
  print(i)
```

実行結果

```
0
1
2
3
4
```

```
# range 関数
# 引数が 1 つの場合，0 からその数値（-1）までの整数を生成
list(range(5))
```

実行結果

```
[0, 1, 2, 3, 4]
```

```
for i in range(5):
  print(i)
```

実行結果

```
0
1
2
3
4
```

# 初級編 第9講 確認問題

range 関数を用いて，次のプログラムと同じ実行結果になるようなプログラムを作成した。作成したプログラムとして最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
for i in [2, 4, 6, 8]:
  print(i)
```

①

```
for i in range(2, 10, 1):
  print(i)
```

②

```
for i in range(2, 9, 2):
  print(i)
```

③

```
for i in range(2, 8, 2):
  print(i)
```

④

```
for i in range(1, 9, 2):
  print(i)
```

初級編

# 第10講 if文

## 要点整理

### if文

条件式が True であれば，インデントされたブロック内の処理を実行する制御構文。

True: 条件式が成り立つ

False: 条件式が成り立たない

```
if 条件式：
  条件式が True の場合に実行する処理
```

```
x = 11
if x > 10:
  print(x, 'は10より大きい')
```

実行結果

```
11 は10より大きい
```

# 初級編　第 10 講　確認問題

次のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
x = 3
if x > 10:
  print(x, 'は10より大きい')
```

① 「3 は 10 より大きい」と表示される

② 「3 は 10 より小さい」と表示される

③ 何も表示されない

④ エラーが発生する

初級編

# 第 11 講 if-else 文

## 要点整理

### if-else 文

条件式が True であれば if のインデントされたブロック内の処理を実行し，False であれば else のインデントされたブロックの処理を実行する制御構文。

```
if 条件式 :
  条件が True の場合に実行する処理
else:
  条件が False の場合に実行する処理
```

```
x = 5
if x > 10:
  print(x, 'は 10 より大きい ')
else:
  print(x, 'は 10 以下 ')
```

実行結果

```
5 は 10 以下
```

# 初級編　第 11 講　確認問題

次のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
x = 10
if x < 10:
  print(x, 'は 10 より小さい')
else:
  print(x, 'は 10 以上')
```

① 「10 は 10 以上」と表示される

② 「10 は 10 より小さい」と表示される

③ 何も表示されない

④ エラーが発生する

初級編

# 第 12 講 if-elif-else 文

## 要点整理

### if-elif-else 文

複数の条件を順番に評価し，最初に True となる条件のインデントされたブロック内の処理を実行する制御構文。

```
if 条件式1:
  条件式1がTrueの場合に実行する処理
elif 条件式2:
  条件式1がFalseで，条件式2がTrueの場合に実行する処理
else:
  すべての条件がFalseの場合に実行する処理
```

```
x = 12
if x > 10:
  print(x, 'は10より大きい')
elif x == 10:
  print(x, 'は10に等しい')
else:
  print(x, 'は10より小さい')
```

実行結果

```
12 は10より大きい
```

# 初級編　第12講　確認問題

次のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
x = 10
if x < 10:
  print(x, 'は10より小さい')
elif x == 10:
  print(x, 'は10に等しい')
else:
  print(x, 'は10より大きい')
```

①　何も表示されない

②　「10 は 10 より大きい」と表示される

③　「10 は 10 より小さい」と表示される

④　「10 は 10 に等しい」と表示される

初級編

# 第 13 講　while 文

## 要点整理

### while 文

while 文は，繰り返し ( ループ ) 処理を行うための制御文。

```
while 条件式 :
  条件式が True の間実行する処理
```

while 文は，指定された条件式が True（成立する）である間，インデントされたブロック内の処理を繰り返し実行する。

条件式が False（成立しない）と判定されると，ループは終了する。

```
# 1 から 5 までの数字を出力
count = 1
while count <= 5:
  print(count)
  count = count + 1
```

実行結果

```
1
2
3
4
5
```

# 初級編　第 13 講　確認問題

次のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
count = 1
while count <= 3:
  print('Hello')
  count = count + 1
```

① 「Hello」が無限に出力される

② 「Hello」が 3 回出力される

③ 「Hello」が 2 回出力される

④ 「Hello」が 1 回出力される

初級編

# 第 14 講 論理演算子

## 要点整理

### 論理演算子

論理積 (and)：and の前後が True であれば True

論理和 (or)：or の前後のどちらか片方でも True であれば True

否定 (not)：論理値（True・False）を反転

```
# 論理積（and）
a = True
b = False
result = a and b  # 1 × 0 と考えるとよい
print(result)  # 0
```

実行結果

```
False
```

```
# 論理和（or）
a = True
b = False
result = a or b  # 1 + 0 と考えるとよい
print(result)  # 1
```

実行結果

```
True
```

```
# 否定（not）
a = True
result = not(a)
print(result)  # 0
```

実行結果

```
False
```

# 初級編　第 14 講　確認問題

次のプログラムにおける，変数 result の値として，最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
a = True
b = True
result = a or b
```

① True

② False

③ None

④ 0

中級編

# 第 15 講 関数

## 例題 1

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def double(x):
  y = 2 * x
  return y

double(10)
```

① 10　② 20　③ 30　④ 40

## 例題 2

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def add(a, b):
  x = a + b
  return x

add(1, 2)
```

① 2　② 3　③ 4　④ 5

# 中級編 第 15 講 確認問題

以下のプログラムの実行結果として最も適切なものを，

あとの①～④のうちから一つ選びなさい。

```
def multiply(a, b):
  return a * b

multiply(3, 5)
```

① 8　② 15　③ -2　④ 2

中級編

# 第16講 配列と乱数

## 例題

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

Janken = ['グー', 'チョキ', 'パー']
number = random.randint(0,2)
print(Janken[number])
```

① エラーになる

② 「0」「1」「2」のいずれかが出力される

③ 「グー」「チョキ」「パー」のいずれかが出力される

④ 「random」が出力される

# 中級編　第 16 講　確認問題

以下のプログラムを実行すると「大吉」「中吉」「小吉」「凶」のいずれかが出力される。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

Omikuji = ['大吉', '中吉', '小吉', '凶']
number = random.randint(0,3)
print(        )
```

① Omikuji

② number

③ random

④ Omikuji[number]

中級編

# 第 17 講　2 次元配列

## 例題

以下のプログラムを実行すると，Bob の 2 つめの科目の得点である「88」が表示される。空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Students_and_scores = [
                            ['Alice', 'Bob', 'Charlie'],
                            [85, 92, 100],
                            [90, 88, 95],
                            [78, 81, 76]
                           ]
value = Students_and_scores[2][□]
print(value)
```

①　0　　②　1　　③　2　　④　'Bob'

# 中級編　第 17 講　確認問題

以下のプログラムを実行すると，Alice の 3 つめの科目の得点である「78」が表示される。空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Students_and_scores = [
                         ['Alice', 'Bob', 'Charlie'],
                         [85, 92, 100],
                         [90, 88, 95],
                         [78, 81, 76]
                        ]
value = □
print(value)
```

① `Students_and_scores[3][1]`

② `Students_and_scores[2][1]`

③ `Students_and_scores[3][0]`

④ `Students_and_scores[2][0]`

中級編

# 第 18 講 input と random.choice

## 例題

以下は，コンピュータとじゃんけんができるプログラムである。ここでは，コンピュータは「グー」「チョキ」「パー」の 3 つからランダムに選んで出力するようにしたい。空欄に入る最も適切なものを，あとの①〜④のうちから一つ選びなさい。

```
import random

user_choice = input('グー，チョキ，パーのどれかを入力：')

Computer_choices = ['グー', 'チョキ', 'パー']
computer_choice = □(Computer_choices)

print('あなたの手：', user_choice)
print('コンピュータの手：', computer_choice)
```

① random.choice

② random.randint

③ random.randrange

④ random.sample

# 中級編 第18講 確認問題

以下のプログラムは，利用者とコンピュータがそれぞれ好きな形を選んで出力するものである。ここでは，コンピュータは「○」「△」「×」の3つからランダムに選んで出力するようにしたい。空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

user_obj = input('〇, △, ×の中から好きな形を入力して:')

Obj = ['〇', '△', '×']
computer_obj = random.choice(□)

print('あなたは', user_obj, 'を選びました。')
print('コンピュータは', computer_obj, 'を選びました。')
```

① user_obj

② Obj

③ computer_obj

④ random

中級編
# 第 19 講 引数が配列の関数

## 例題

以下のプログラムを実行すると，Numbers = [85, 92, 100] の要素の合計が表示される。calculate_sum 関数が配列の合計を計算して返すように，空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def calculate_sum(Arr):
  total = 0
  for num in □ :
    total = total + num
  return total

Numbers = [85, 92, 100]
calculate_sum(Numbers)
```

① Arr

② total

③ numbers

④ result

# 中級編　第 19 講　確認問題

以下のプログラムを実行すると，Numbers = [12, 14, 15, 21, 25] の要素の合計が表示される。
空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def calculate_sum(Arr):
  total = 0
  for num in Arr:
    total = total + num
  return total

Numbers = [12, 14, 15, 21, 25]
result = □
print(result)
```

① calculate_sum(Numbers)

② total

③ calculate_sum(total)

④ Numbers

中級編

# 第20講 1からnまでの和

## 例題

以下のプログラムを実行すると，1からnまでの合計（例えば，n = 5の場合は1+2+3+4+5 = 15）が表示される。

calculate_sum関数が1からnまでの合計を計算して返すように，空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def calculate_sum(n):
  total = 0
  for i in [    ]:
    total = total + i
  return total

n = 5
calculate_sum(n)
```

① `range(1, n)`

② `range(1, n + 1)`

③ `range(0, n)`

④ `range(0, n - 1)`

# 中級編　第 20 講　確認問題

以下のプログラムを実行すると，1 から n までの合計（例えば, n = 5 の場合は 1+2+3+4+5 = 15）が表示される。calculate_sum 関数が 1 から n までの合計を計算して返すように，空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def calculate_sum(n):
  total = 0
  for i in range(1, n + 1):
    total = □
  return total

n = 5
result = calculate_sum(n)
print(result)
```

① i

② total

③ total - i

④ total + i

中級編

# 第 21 講 ○以上○以下の判定

## 例題

以下のプログラムは, 身長（cm）h の値によって, 150 未満なら S, 150 以上 160 未満なら M, 160 以上なら L と表示する check_size 関数である。

空欄ア, イに入る最も適切なものの組み合わせを, あとの①～④のうちから一つ選びなさい。

```
def check_size(h):
  if [  ア  ]:
    return 'S'
  elif [  イ  ]:
    return 'M'
  else:
    return 'L'

h = 145
check_size(h)
```

① ア : h < 150　イ : h < 160

② ア : h < 160　イ : h >= 150

③ ア : h < 150　イ : h >= 160

④ ア : h < 160　イ : h < 150

# 中級編 第 21 講 確認問題

以下のプログラムは, テストの得点をscoreに代入すると, 60 以上なら A, 50 以上 60 未満なら B, それ以外なら C と表示するものである。空欄ア, イに入る最も適切なものの組み合わせを, あとの①～④のうちから一つ選びなさい。

```
def check_grade(score):
  if [ ア ]:
    return 'A'
  elif [ イ ]:
    return 'B'
  else:
    return 'C'

score = 55
print(check_grade(score))
```

① ア : `score >= 60`　イ : `score >= 50`

② ア : `score >= 50`　イ : `score >= 60`

③ ア : `score >= 50`　イ : `score < 60`

④ ア : `score > 60`　イ : `score >= 50 and score <= 60`

中級編

# 第 22 講　2 つの数字の並べ替え

## 例題

以下のプログラムは，配列 Arr の要素（2 つの整数）を昇順（小さい順）に並べ替える関数である。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def sort_Arr(Arr):
  if □:
    temp = Arr[0]
    Arr[0] = Arr[1]
    Arr[1] = temp
  return Arr

Arr1 = [5, 2]
sort_Arr(Arr1)
```

①　Arr[0] < Arr[1]

②　Arr[0] > Arr[1]

③　Arr1[1] > Arr1[0]

④　Arr1[1] < Arr1[0]

# 中級編　第22講　確認問題

以下のプログラムは，２つの数 a と b を大きい順に並べ替えるものである。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def sort_two_numbers(a, b):
  if □:
    temp = a
    a = b
    b = temp
  return a, b

result = sort_two_numbers(2, 5)
print(result)
```

実行結果

```
(5, 2)
```

① a < b

② a > b

③ 2 < 5

④ 2 > 5

中級編

# 第 23 講　合計と平均

## 例題

以下のプログラムは, 配列 Data = [10, 20, 30, 40, 50] の要素の平均を求めるものである。空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Data = [10, 20, 30, 40, 50]
total = 0

for num in Data:
  total = total + num

data_average = □
print(data_average)
```

① `total / len(Data)`

② `total / num`

③ `len(Data)`

④ `Data[num]`

# 中級編 第23講 確認問題

以下のプログラムは，配列 Data = [10, 20, 30, 40, 50] の要素の合計を求めるものである。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Data = [10, 20, 30, 40, 50]
total = 0

for num in Data:
  total = □

print(total)
```

① total + num

② total * num

③ Data[num]

④ num

中級編

# 第24講 for 文の入れ子

## 例題

以下のプログラムは，n の段までのかけ算九九の結果を表示するものである。

例えば，n = 3 ならば，「1 × 1 = 1」「1 × 2 = 2」…「3 × 9 = 27」を表示する。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def kakezan(n):
  for i in □:
    for j in range(1, 10):
      value = i * j
      print(i, '×', j, '=', value)

n = 3
kakezan(n)
```

① range(n)

② range(0, n)

③ range(1, n)

④ range(1, n + 1)

# 中級編 第 24 講 確認問題

以下のプログラムは，かけ算九九の結果を「1 × 1 = 1」から「9 × 9 = 81」まで全て表示するものである。
2 つの空欄に共通して入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
for i in □:
  for j in □:
    value = i * j
    print(i, ' × ', j, ' = ', value)
```

① range(0, 9)

② range(1, 9)

③ range(1, 10)

④ range(2, 10)

中級編

# 第25講 for 文と if 文の入れ子

## 例題

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
for i in range(3):
  for j in range(2):
    if i == j:
      print('A')
    else:
      print('B')
```

| ① | ② | ③ | ④ |
|---|---|---|---|
| A | A | A | B |
| B | B | B | A |
| B | A | B | B |
| B | B | A | A |
| B | B | B | B |
| B | B | B | B |

# 中級編　第 25 講　確認問題

以下のプログラムを実行すると，1 から n までの奇数の合計（例えば, n ＝ 5 の場合は 1+3+5 ＝ 9）が表示される。calculate_odd_sum 関数が 1 から n までの奇数の合計を計算して返すように，空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def calculate_odd_sum(n):
  total = 0
  for i in range(1, n + 1):
    if i % 2 == 1:
      total = total + [    ]
  return total

n = 5
result = calculate_odd_sum(n)
print(result)
```

① `i`

② `i + 1`

③ `n`

④ `total`

中級編

# 第 26 講　最大値と最小値

## 例題

以下のプログラムは，与えられた配列の要素から最小値を求めるものである。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def find_min(Numbers):
  min_value = Numbers[0]
  for num in Numbers:
    if □:
      min_value = num

  return min_value

Numbers = [90, 80, 55, 77, 92]
find_min(Numbers)
```

① num > min_value

② num < min_value

③ num == min_value

④ num >= min_value

# 中級編　第 26 講　確認問題

以下のプログラムは，与えられた配列の要素から最大値を求めるものである。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
def find_max(Numbers):
  max_value = Numbers[0]
  for num in Numbers:
    if □:
      max_value = num

  return max_value

Numbers = [90, 80, 55, 77, 92]
max_value = find_max(Numbers)
print(max_value)
```

① num > max_value

② num < max_value

③ num == max_value

④ num <= max_value

中級編

# 第27講 カウントダウン

## 例題

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
count = 10
while count > 5:
  print(count)
  count = count - 1
```

| ① | ② | ③ | ④ |
|---|---|---|---|
| 9<br>8<br>7<br>6<br>5 | 9<br>8<br>7<br>6 | 10<br>9<br>8<br>7<br>6 | 10<br>9<br>8<br>7<br>6<br>5 |

# 中級編　第 27 講　確認問題

以下のプログラムとその実行結果をもとに，空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
count = 5
while count > 0:
  print(count)
  count = □
```

実行結果

```
5
4
3
2
1
```

① count + 1

② count - 1

③ count

④ 1 - count

中級編

# 第28講 while 文と if 文の入れ子

## 例題

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
i = 1
while i <= 10:
  if i % 3 == 0:
    print('Oh!')
  else:
    print(i)
  i = i + 1
```

| ① | ② | ③ | ④ |
|---|---|---|---|
| Oh! | 1 | 1 | 1 |
| Oh! | 2 | 2 | 2 |
| Oh! | 3 | Oh! | Oh! |
| Oh! | 4 | 4 | 3 |
| Oh! | 5 | 5 | 4 |
| Oh! | 6 | Oh! | 5 |
| Oh! | 7 | 7 | Oh! |
| Oh! | 8 | 8 | 6 |
| Oh! | 9 | Oh! | 7 |
| Oh! | 10 | 10 | 8 |
| | | | Oh! |
| | | | 9 |
| | | | 10 |

# 中級編 第28講 確認問題

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
x = 0
while x < 5:
  if x == 3:
    print('Three')
  else:
    print(x)
  x = x + 1
```

| ① | ② | ③ | ④ |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 2 | 1 | 2 |
| 2 | 3 | 2 | Three |
| Three | 4 | 3 | 4 |
| 4 | 5 | 4 | 5 |

中級編

# 第 29 講　10 進法から 2 進法への変換

## 例題

以下のプログラムは，10 進法の整数を，対応する 2 進法の文字列に変換するものである。例えば，変数 num に 10 を代入してプログラムを実行すると，2 進法の文字列「1010」が出力される。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
num = 10
binary = ''
while num > 0:
  remainder = □
  binary = str(remainder) + binary
  num = num // 2

print(binary)
```

実行結果

```
1010
```

① num / 2

② num // 2

③ num * 2

④ num % 2

# 中級編　第29講　確認問題

以下のプログラムは，10進法の整数を2進法の文字列に変換するものである。
例えば，変数numに26を代入してプログラムを実行すると，2進法の文字列「11010」が出力される。
空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
num = 26
binary = ''
while num > 0:
  remainder = num % 2
  binary = str(remainder) + binary
  num = □

print(binary)
```

実行結果

```
11010
```

① num / 2

② num // 2

③ num * 2

④ num % 2

中級編

# 第30講 ソート（並べ替え）

## 例題

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Numbers = [3, 1, 4, 1, 5]
for i in range(len(Numbers)):
  for j in range(len(Numbers) - i - 1):
    if Numbers[j] > Numbers[j + 1]:
      temp = Numbers[j]
      Numbers[j] = Numbers[j + 1]
      Numbers[j + 1] = temp
print(Numbers)
```

① [5, 1, 4, 1, 3]

② [5, 3, 1, 1, 4]

③ [1, 3, 4, 5, 1]

④ [1, 1, 3, 4, 5]

# 中級編　第 30 講　確認問題

以下のプログラムは，配列 Numbers の要素を降順（大きい順）に並べ替えるものである。

空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Numbers = [3, 1, 4, 1, 5]
for i in range(len(Numbers)):
  for j in range(0, len(Numbers) - i - 1):
    if ▭ :
      temp = Numbers[j]
      Numbers[j] = Numbers[j + 1]
      Numbers[j + 1] = temp
print(Numbers)
```

① Numbers[j - 1] > Numbers[j]

② Numbers[j - 1] < Numbers[j]

③ Numbers[j] > Numbers[j + 1]

④ Numbers[j] < Numbers[j + 1]

中級編

# 第 31 講 配列への要素の追加

## 例題

配列 Numbers = [1, 2, 3] に要素 4 を追加して，配列 Numbers が [1, 2, 3, 4] になるようにしたい。

最も適切なプログラムを，あとの①～④のうちから一つ選びなさい。

① `Numbers + 4`

② `Numbers.add(4)`

③ `Numbers.append(4)`

④ `Numbers.append[4]`

# 中級編 第 31 講 確認問題

配列 Subjects = ['国語', '数学'] に要素 '英語' を追加するプログラムとして最も適切なものを，あとの①～④のうちから一つ選びなさい。

① `Subjects.add('英語')`

② `Subjects.append(英語)`

③ `Subjects.append('英語')`

④ `Subjects.insert('英語')`

中級編

# 第 32 講　2 次元配列の応用

## 例題

以下のプログラムと実行結果をもとに，空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Score = [
          ['国語', '数学', '英語'],
          [85, 92, 100]
        ]
num = len(□)
for i in range(num):
  print(Score[0][i], 'の点数は' , Score[1][i], '点')
```

実行結果

```
国語 の点数は 85 点
数学 の点数は 92 点
英語 の点数は 100 点
```

① Score

② Score[0]

③ Score[1][0]

④ Score[0][1]

# 中級編　第 32 講　確認問題

以下のプログラムと実行結果をもとに，空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Fruits_and_numbers = [
                        ['りんご', 'みかん', 'バナナ'],
                        [10, 12, 18]
                       ]
num = len(      )
for i in range(num):
  print(Fruits_and_numbers[1][i], '個の', Fruits_and_numbers[0][i])
```

実行結果

```
10 個の りんご
12 個の みかん
18 個の バナナ
```

① num

② Fruits_and_numbers

③ Fruits_and_numbers[1]

④ Fruits_and_numbers[1][0]

中級編
# 第33講 辞書型

## 例題

以下のプログラムの実行結果として最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
person = {
  'name': '太郎',
  'age': 20,
  'country': '日本'
}

print(person['name'])
```

① 「20」と表示される

② 「太郎」と表示される

③ 「日本」と表示される

④ エラーとなる

# 中級編　第 33 講　確認問題

以下のプログラムの実行結果として最も適切なものを，

あとの①～④のうちから一つ選びなさい。

```
fruits = {
  'apple': 100,
  'banana': 200,
  'cherry': 300
}

total = fruits['apple'] + fruits['cherry']
print(total)
```

①　300　　②　400　　③　500　　④　600

上級編

# 第34講 コインの枚数

## 例題

銀行のATMである金額を引き出すとき，紙幣・硬貨の枚数の合計が最も少なくなるように紙幣・硬貨がそれぞれ何枚必要かを表示するプログラムを作成する。

例えば，12,346円を引き出す場合は以下のように表示される。

実行結果

```
10000円：1枚
5000円：0枚
1000円：2枚
500円：0枚
100円：3枚
50円：0枚
10円：4枚
5円：1枚
1円：1枚
```

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Okane = [10000, 5000, 1000, 500, 100, 50, 10, 5, 1]
Maisuu = [0, 0, 0, 0, 0, 0, 0, 0, 0] # 引き出すお金の枚数を保持（Okane配列の添字と対応）
okane_count = 0 # お金の枚数を一時的に保持

hikidashi = int(input('引き出し金額を入力してください：'))

for i in range(len(Okane)):
  okane_count = □
  Maisuu[i] = okane_count
  hikidashi = hikidashi % Okane[i]

# 表示処理
for i in range(len(Okane)):
  print(Okane[i],'円：',Maisuu[i],'枚')
```

① `hikidashi * Okane[i]`

② `hikidashi % Okane[i]`

③ `hikidashi ^ Okane[i]`

④ `hikidashi // Okane[i]`

# 上級編　第 34 講　確認問題

以下の条件で予算を入力すると，いくらのお菓子を何個買えるか表示するプログラムを作成する。

- 350 円，170 円，90 円，55 円，30 円，20 円の 6 種類のお菓子が売っている商店で買い物をする。
- できるだけ高いお菓子をたくさん買い，予算をできるだけ使い切る買い方をする。
- 配列 Okashi には，お菓子の値段が降順に格納されている。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Okashi = [350, 170, 90, 55, 30, 20] # 売っているお菓子の金額 ( 値段の降順 )
Kosuu = [0, 0, 0, 0, 0, 0]

kingaku = int(input(' 予算を入力してください：'))

for i in range(len(Okashi)):
  Kosuu[i] = kingaku // Okashi[i]
  kingaku = [            ]

for i in range(len(Okashi)):
  print(Okashi[i],' 円のお菓子：',Kosuu[i],' 個 ')
```

① `Kosuu[i] * kingaku`

② `kingaku * Okashi[i]`

③ `Okashi[i] % kingaku`

④ `kingaku % Okashi[i]`

上級編

# 第 35 講 駐車料金の判定

## 例題

以下の条件で，駐車場の料金計算を行うプログラムを作成する。

・料金設定は，最初の 2 時間までの基本料金が 500 円，それ以降は 1 時間ごとに 400 円である。

・例えば，5 時間駐車した場合は，最初の 2 時間は 500 円，その後の 3 時間は 1,200 円となるので合計した駐車料金は 1,700 円である。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
kihon_ryoukin = 500 # 基本料金
jikan_ryoukin = 400 # 基本時間終了後の 1 時間ごとの金額
kihon_jikan = 2 # 基本時間（最初の 2 時間は基本料金のみ）
shiharai_ryoukin = 0

parked_jikan = int(input('駐車した時間を整数値で入力してください（例：4 時間の場合は 4 と入力）: '))

if parked_jikan <= kihon_jikan:
  shiharai_ryoukin = kihon_ryoukin
else:
  shiharai_ryoukin = [　　　　　　　　]

print('駐車料金', shiharai_ryoukin, '円')
```

① kihon_ryoukin + parked_jikan - kihon_jikan * jikan_ryoukin

② parked_jikan * jikan_ryoukin

③ (parked_jikan - kihon_jikan) * jikan_ryoukin

④ kihon_ryoukin + (parked_jikan - kihon_jikan) * jikan_ryoukin

# 上級編　第 35 講　確認問題

以下の条件で，タクシーの料金計算を行うプログラムを作成する。

・料金設定は, 初乗り距離 : 2km まで 500 円, それ以降 : 1km ごとに 300 円である。

・例えば, 5km 利用した場合は, 最初の 2km は 500 円, その後の 3km は 900 円となるので合計した乗車料金は 1,400 円である。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
hatsunori_ryoukin = 500 # 初乗り料金
kiro_ryoukin = 300 # 1km ごとの料金
hatsunori_kyori = 2 # 初乗り距離（初乗り料金のみの距離 :km）
jousya_ryoukin = 0

jousya_kyori = int(input(' 乗車した距離を整数値で入力してください ( 例 : 6km の場合は 6 と入力 ): '))

if jousya_kyori > hatsunori_kyori:
  jousya_ryoukin = [          ]
else:
  jousya_ryoukin = hatsunori_ryoukin

print(' 乗車料金 ', jousya_ryoukin, ' 円 ')
```

① jousya_kyori - hatsunori_kyori * kiro_ryoukin + hatsunori_ryoukin

② (jousya_kyori + hatsunori_kyori) * kiro_ryoukin + hatsunori_ryoukin

③ (jousya_kyori - hatsunori_kyori) * kiro_ryoukin + hatsunori_ryoukin

④ (jousya_kyori + hatsunori_kyori) / kiro_ryoukin * hatsunori_ryoukin

上級編

# 第 36 講 割引のある料金

## 例題

以下の条件で，ある駄菓子屋で買い物をした際に割引率を加味した合計金額を表示するプログラムを作成する。

- この駄菓子屋では，買ったお菓子の個数によって割引が行われる。
- 3 個または 4 個買った場合は 5%，5 個以上買った場合は 10% を合計金額から割引きする。
- 購入するお菓子の値段は Okashi 配列に格納されており，配列の要素数が購入数と一致する。
- sum 関数は，引数で与えられた配列の要素を合計する関数である。
- len 関数は，引数で与えられた配列の要素数を求める関数である。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Okashi= [140, 160, 170, 140]

goukei_kakaku = sum(Okashi)
okashi_kosuu = len(Okashi)
waribiki_ritsu = 0.00

if okashi_kosuu == 3 or okashi_kosuu == 4:
  waribiki_ritsu = 0.05
elif okashi_kosuu >= 5:
  waribiki_ritsu = 0.10
else:
  waribiki_ritsu = 0.00

waribiki_kakaku = [          ]

print(' 合計金額 :', int(waribiki_kakaku), ' 円 ')
```

① goukei_kakaku * waribiki_ritsu

② goukei_kakaku * 1 - waribiki_ritsu

③ goukei_kakaku / (1 - waribiki_ritsu)

④ goukei_kakaku * (1 - waribiki_ritsu)

＜MEMO＞

# 上級編　第 36 講　確認問題

以下の条件で，ある電力会社の割引率を加味した電気利用料金を計算するプログラムを作成する。

- 料金設定は，基本料金が 500 円，従量利用料金が電気の利用 1kWh ごとに 100 円である。
- 電気の利用量に応じて割引率を設けており，100kWh 以上 200kWh 未満の場合は 10%，200kWh 以上の場合は 15% を合計金額から割引する。
- 例えば，電気の利用量が 200kWh だった場合は，基本料金 500 円と従量利用料金 20,000 円の合計 20,500 円から 15% を割引した 17,425 円が電気利用料金となる。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
kihon_ryoukin = 500 # 基本料金
ryoukin_kwh = 100 # 1kWh あたりの価格
discount_rate = 0.00

denryoku = int(input('使用した電力量 (kWh) を整数値で入力してください ( 例 : 200kWh の場合は 200 と入力 ): '))

if denryoku < 100:
  discount_rate = 0.00
elif denryoku < 200:
  discount_rate = 0.10
else:
  discount_rate = 0.15

denki_ryoukin = 

print('電気料金：', int(denki_ryoukin), '円')
```

① (kihon_ryoukin + ryoukin_kwh * denryoku) * (1 - discount_rate)

② (kihon_ryoukin + ryoukin_kwh * denryoku) * discount_rate

③ kihon_ryoukin + ryoukin_kwh * denryoku * (1 - discount_rate)

④ (kihon_ryoukin + ryoukin_kwh * denryoku) * (discount_rate - 1)

上級編

# 第 37 講 テストの集計

## 例題

以下の条件で，複数のテストの得点の合計点と平均点を計算するプログラムを作成する。

・配列 Tensuu に集計対象のテストの得点が格納されている。

・平均点は round 関数を使って，小数点以下第 2 位を四捨五入する。

・round 関数の第 1 引数に四捨五入したい値，第 2 引数に四捨五入後の小数点以下の表示桁数を指定する。（例：round(65.78,1)=65.8）

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Tensuu = [85, 90, 78, 88, 76, 94, 89, 75, 83]

num = len(Tensuu)
goukei = 0
heikin = 0

for i in range(num):
  goukei = goukei + Tensuu[i]

heikin = [            ] # 小数点以下第 2 位を四捨五入
print(' 合計点 :', goukei, ' 点 ')
print(' 平均点 :', heikin, ' 点 ')
```

① round(goukei/num, 1)

② round(goukei/num, 2)

③ round(count/goukei, 1)

④ round(goukei, 2)

# 上級編 第 37 講 確認問題

複数のテストの得点の合計点と最高点を表示するプログラムを作成する。

なお，配列 Tensuu には集計対象のテストの得点が格納されている。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Tensuu = [85, 90, 78, 88, 76, 94, 89, 75, 83]

num = len(Tensuu)
goukei = 0
saikou = 0

for i in range(num):
  goukei = goukei + Tensuu[i]
  if ＿＿＿＿＿＿ :
    saikou = Tensuu[i]

print('合計点:', goukei, '点')
print('最高点:', saikou, '点')
```

① `Tensuu[i] < saikou`

② `Tensuu[i] > saikou`

③ `Tensuu[i] > goukei`

④ `Tensuu[i] == saikou`

上級編

# 第 38 講 じゃんけん

## 例題

以下の条件で，コンピュータとじゃんけんを行うプログラムを作成する。

・自分自身の手は入力し，相手の手はランダムに出される。

・じゃんけんの「勝ち」「負け」「あいこ」のパターンについて表にすると以下のようになる。

| | | 相手側 | | |
|---|---|---|---|---|
| | | グー | チョキ | パー |
| 自分側 | グー | あいこ | 勝ち | 負け |
| | チョキ | 負け | あいこ | 勝ち |
| | パー | 勝ち | 負け | あいこ |

・グーを 0，チョキを 1，パーを 2 の数字に置き換えて，自分の手の数字から相手の手の数字を引いた結果は以下のようになる。

| | | 相手側 | | |
|---|---|---|---|---|
| | | グー :0 | チョキ :1 | パー :2 |
| 自分側 | グー :0 | 0 | -1 | -2 |
| | チョキ :1 | 1 | 0 | -1 |
| | パー :2 | 2 | 1 | 0 |

・上記の数字を活用してプログラムを作成する。

・変数 jibun には自分側の手に該当する数字を入力し，変数 aite には相手側（コンピュータ）の手に該当する数字をランダムに代入する。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

Janken = ['グー', 'チョキ', 'パー']
print('ジャンケンの手の数字を入力してください。グー：0，チョキ：1，パー：2')
jibun = int(input())
aite = random.randint(0,2)
keisan = jibun - aite

if [          ]:
  print('勝ち')
elif keisan == 1 or keisan == -2:
  print('負け')
else:
  print('あいこ')

print('自分の手：', Janken[jibun], '相手の手：', Janken[aite])
```

① keisan == -1 or keisan == 2

② keisan == -1 and keisan == 2

③ keisan == 1 or keisan == -2

④ keisan == 0

# 上級編　第 38 講　確認問題

以下の条件で，コンピュータとじゃんけんを行うプログラムを作成する。

・自分自身の手は入力し，相手の手はランダムに出される。

・じゃんけんの「勝ち」「負け」「あいこ」のパターンについて表にすると以下のようになる。

| | | 相手側 | | |
|---|---|---|---|---|
| | | グー | チョキ | パー |
| 自分側 | グー | あいこ | 勝ち | 負け |
| | チョキ | 負け | あいこ | 勝ち |
| | パー | 勝ち | 負け | あいこ |

・グーを 0，チョキを 1，パーを 2 の数字に置き換えてプログラムを作成する。

・自分の手の数字と相手の手の数字を使ってある計算を行うと，その結果は以下のようになり，プログラムではこの値を変数 keisan に代入して活用する。

| | | 相手側 | | |
|---|---|---|---|---|
| | | グー :0 | チョキ :1 | パー :2 |
| 自分側 | グー :0 | 0 | 2 | 1 |
| | チョキ :1 | 1 | 0 | 2 |
| | パー :2 | 2 | 1 | 0 |

・変数 jibun には自分側の手に該当する数字を入力し，変数 aite には相手側（コンピュータ）の手に該当する数字をランダムに代入する。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

Janken = ['グー', 'チョキ', 'パー']
print('出す手の数字を入力してください。グー:0, チョキ:1, パー：2')
jibun = int(input())
aite = random.randint(0,2)
keisan = [          ]

if keisan == 2:
  print('あなたの勝ち')
elif keisan == 1:
  print('あなたの負け')
else:
  print('引き分け')

print('自分の手：', Janken[jibun], '相手の手：', Janken[aite])
```

① jibun - aite

② jibun - aite + 3

③ (jibun - aite + 3) % 3

④ (jibun - aite + 2) % 3

上級編

# 第 39 講　サイコロの出目

## 例題

六面のサイコロをある回数振った場合，それぞれの目が出る確率がどのようになるかを検証するシミュレーションプログラムを作成する。

例えば，サイコロを 100,000 回振る場合のシミュレーションを行うと以下のように表示される。

実行結果

```
出目 1 : 16727 回　確率 : 16.727 %
出目 2 : 16497 回　確率 : 16.497 %
出目 3 : 16912 回　確率 : 16.912 %
出目 4 : 16591 回　確率 : 16.591 %
出目 5 : 16588 回　確率 : 16.588 %
出目 6 : 16685 回　確率 : 16.685 %
```

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

kaisuu = 100000 # サイコロを振る回数
Deme_kaisuu = [0, 0, 0, 0, 0, 0] # 出目の回数を格納する
kakuritsu = 0 # 各出目が出る確率を格納する

for i in range(kaisuu) :
  deme = random.randint(1, 6)
  [                              ]

# 結果を表示
for i in range(6):
  kakuritsu = Deme_kaisuu[i] / kaisuu * 100
  print(' 出目 ', i + 1, ':', Deme_kaisuu[i], ' 回 ',  ' 確率 :', kakuritsu, '% ')
```

① Deme_kaisuu[deme] = Deme_kaisuu[deme] + 1

② Deme_kaisuu[deme - 1] = Deme_kaisuu[deme - 1]

③ Deme_kaisuu[deme - 1] = Deme_kaisuu[deme - 1] + 1

④ Deme_kaisuu[deme - 1] = Deme_kaisuu[deme - 1] - 1

# 上級編　第 39 講　確認問題

六面のサイコロを２個同時に振ったとき，出目の合計が８になる確率を検証するシミュレーションプログラムを作成する。
例えば，100,000 回試行するシミュレーションを行うと，以下のように表示される。

実行結果
```
出目の合計 8 : 13831 回　確率 : 13.831 %
```

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

kaisuu = 100000 # サイコロを振る回数
deme_kaisuu = 0 # 出目の回数を格納する
goukei = 8 # 確率を求めたい合計値（今回は８）
kakuritsu = 0 # 出目の合計値が出る確率を格納する

for i in range(kaisuu):
  deme1 = random.randint(1, 6)
  deme2 = random.randint(1, 6)
  if (deme1 + deme2) == goukei:
    deme_kaisuu = deme_kaisuu + 1

[　　　　　　]

# 結果を表示
print('出目の合計', goukei, ':', deme_kaisuu, '回', '確率:', kakuritsu, '%')
```

① kakuritsu = deme_kaisuu / kaisuu

② kakuritsu = kaisuu / deme_kaisuu * 100

③ deme_kaisuu = kakuritsu / kaisuu * 100

④ kakuritsu = deme_kaisuu / kaisuu * 100

上級編

# 第 40 講　中央値の判定

## 例題

以下の条件で，中央値を求めるプログラムを作成する。

- 例えば, 昇順に並べられた配列 1,3,6,7,9 の場合の中央値はちょうど真ん中の 6 となる。
- 配列 1,3,7,9 の場合は，ちょうど真ん中は 3 と 7 の間なので 3 と 7 の平均を求めて 5 となる。
- 「//」は，商の整数値を求める演算子である。
- 「%」は，余りを求める演算子である。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Data = [1, 3, 7, 9] # 昇順のデータ配列
center = 0 # 算出した中央値を格納
n = len(Data) # nに要素数を格納

if n % 2 == 0:
  center = (Data[n // 2 - 1] + Data[n // 2]) / 2
else:
  center = [          ]

print(' 中央値：', int(center))
```

① Data[n // 2]

② Data[n % 2]

③ Data[n // 2] / 2

④ Data[n // 2 + 1] + Data[n // 2]

＜MEMO＞

# 上級編　第 40 講　確認問題

以下の条件で，第 1 四分位数を求めるプログラムを作成する。

- 第 1 四分位数とは，データを小さい順に並べたとき，初めから数えて 25% の位置にある数のことである。
- 例えば, 昇順に並べられたデータ数が偶数個の配列 [1, 3, 6, 7, 8, 9] の場合は, 1,3,6 と 7,8,9 の 2 つのグループに分け, 1,3,6 の真ん中の 3 が第 1 四分位数になる。
- データ数が奇数個の配列 [1,4,6,8,9,10,13] の場合は，真ん中の 8 を除いて 1,4,6 と 9,10,13 の 2 つのグループに分け, 1,4,6 の真ん中の 4 が第 1 四分位数になる。
- 左側のグループのデータが 1,5,7,9 のように偶数個の場合は，5 と 7 の平均をとって 6 が第 1 四分位数となる。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Data = [1, 4, 6, 8, 9, 10, 13, 15] # 昇順のデータ配列
Left_data = [] # 左側のグループのデータを入れる配列
n = len(Data)
midpoint = n // 2

# for 文を使用して，左側のグループデータを作成
for i in range(0, midpoint):
  Left_data.append(Data[i])

left_n = len(Left_data)

# 左側のデータセットから中央値を取得
if left_n % 2 == 0:
  first_quartile = □
else:
  first_quartile = Left_data[left_n // 2]

print(int(first_quartile))
```

① `Left_data[left_n // 2 + 1] + Left_data[left_n // 2]`

② `Left_data[left_n // 2] / 2`

③ `(Left_data[left_n // 2 - 1] + Left_data[left_n // 2]) / 2`

④ `Left_data[left_n - 1] + Left_data[left_n]`

上級編

# 第 41 講 素数判定

## 例題

入力された自然数（正の整数）が素数かどうかを判定するプログラムを作成する。

なお，素数とは 1 より大きい整数で，1 とその数でしか割り切れない数のことである。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
number = int(input('2 以上の自然数（正の整数）を入力してください。'))
sosuu_flag = 0

for i in range(2, number):
  if [            ]:
    sosuu_flag = 1

# 判定結果を出力
if sosuu_flag == 0:
  print(number, ' は素数です。')
else:
  print(number, ' は素数ではありません。')
```

① `number / i == 0`

② `number % i == 1`

③ `number % i == 0`

④ `number % i != 0`

# 上級編　第 41 講　確認問題

以下の条件で，入力された２つの数字が互いに素であるかを判定するプログラムを作成する。

・「互いに素」とは，２つの整数の共通の約数が１以外にないことである。

・min 関数とは，引数で与えられた数の中で最も小さい数を返す関数である。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
number1 = int(input('1つ目の自然数（正の整数）を入力してください。'))
number2 = int(input('2つ目の自然数（正の整数）を入力してください。'))

smallest = min(number1, number2)
max_gcd = 1 # 互いに素の場合，最大公約数は1

# 2つの数の公約数があるかを調査
for i in range(2,smallest + 1):
  if [　　　　]:
    max_gcd = i

# 2つの数が互いに素であるかを判定
if max_gcd == 1:
  print(number1, 'と', number2, 'は互いに素です。')
else:
  print(number1, 'と', number2, 'は互いに素ではありません。')
  print('最大公約数は', max_gcd, 'です。')
```

① `number1 % i == 0 and number2 % i == 0`

② `number1 % i == 0 or number2 % i == 0`

③ `number1 % i == 0 not number2 % i == 0`

④ `number1 % i != 0 and number2 % i != 0`

上級編

# 第 42 講 素因数分解

## 例題

以下の条件で，整数 number の約数を効率的に算出して列挙するプログラムを作成する。

・約数とは，ある整数を割ることができ商が整数になる数のことである。

・Yakusuu.sort() という命令は，配列 Yakusuu の要素を昇順に並べ替えるものである。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
number = 36
Yakusuu = [] # 求めた約数を格納する配列

for i in range(1, int(number ** 0.5) + 1): # number ** 0.5 は，number の平方根
  if [          ]:
    Yakusuu.append(i) # 約数 i を Yakusuu 配列に追加する
    pair_yakusuu = number // i # 約数 i のペア約数
    if i != pair_yakusuu:
      Yakusuu.append(pair_yakusuu)

Yakusuu.sort() # 配列内の要素を昇順に並べ替える
print(number, 'の約数:', Yakusuu)
```

① `number % i != 0`

② `number % i == 0`

③ `number // i == 1`

④ `number % i == 1`

# 上級編　第 42 講　確認問題

入力された整数を素因数分解し，結果を効率的に列挙するプログラムを作成する。

なお，素因数分解とは，与えられた整数をその数を構成する素数の積に分解することである。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
number = int(input('素因数分解を行う整数を入力してください。'))

original_number = number # number の初期値を保存しておく
i = 2 # 素数の候補を一時的に格納（2 は最小の素数）
Factors = [] # 因数分解した値を格納する配列

while i * i <= number:
  if [          ]:
    i = i + 1
  else:
    number = number // i
    Factors.append(i) # 引数の値を配列 Factors に追加する
if number > 1: # 空の配列にならないための条件
  Factors.append(number)

print(original_number, 'の素因数分解 :', Factors)
```

① `number % i != 0`

② `number % i == 0`

③ `number / i > 0`

④ `number // i == 1`

上級編

# 第 43 講　倍数の判定・FizzBuzz

## 例題

以下の条件で，1 から 100 までの数字を画面に表示するプログラムを作成する。

・5 の倍数かつ偶数のときは，数字の代わりに「five_even」と表示する。

・7 の倍数かつ奇数のときは，数字の代わりに「seven_odd」と表示する。

・いずれの条件にも当てはまらないときは，数字をそのまま表示する。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
for i in range(1, 101):
  if i % 5 == 0 and i % 2 == 0:
    print('five_even')
  elif [          ]:
    print('seven_odd')
  else:
    print(i)
```

① `i % 7 == 0 and i % 2 == 0`

② `i / 7 == 0 or i % 2 == 1`

③ `i % 7 == 1 and i % 2 == 1`

④ `i % 7 == 0 and i % 2 == 1`

# 上級編　第 43 講　確認問題

以下の条件で，1 から 100 までの数字を画面に表示する「FizzBuzz（フィズバズ）問題」のプログラムを作成する。

- 3 の倍数のときは，数字の代わりに「Fizz」と表示する。
- 5 の倍数のときは，数字の代わりに「Buzz」と表示する。
- 3 の倍数かつ 5 の倍数のとき（上記の条件をすべて満たす時）は，数字の代わりに「FizzBuzz」と表示する。
- いずれの条件にも当てはまらないときは，数字をそのまま表示する。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
for i in range(1, 101):
  if [                    ]:
    print('FizzBuzz')
  elif i % 3 == 0:
    print('Fizz')
  elif i % 5 == 0:
    print('Buzz')
  else:
    print(i)
```

① `i % 3 == 0 or i % 5 == 0`

② `i % 3 != 0 and i % 5 != 0`

③ `i / 3 == 0 not i / 5 == 0`

④ `i % 3 == 0 and i % 5 == 0`

上級編

# 第44講 バブルソート・選択ソート

## 例題

以下の条件で，交換法（バブルソート）を使って配列の要素を昇順に並べ替えるプログラムを作成する。

- 交換法（バブルソート）とは，隣り合う二つの要素を比較して，昇順または降順に並べ替えを行う整列アルゴリズムである。
- 今回は，配列の左側から隣り合う二つの要素を比較していき，昇順に並べ替えを行う。
- 例えば，[3,2,4] というデータがある場合，次のような手順で並べ替えを行う。
  - 初めは３と２を比較して，左側の方が大きいので３と２を入れ替えて [2,3,4] となる。
  - 次に３と４を比較して，左側の方が小さいので入れ替えは行わない。
  - これで一番大きな値である４が確定する。
  - 配列の先頭に戻り２と３を比較し，左側の方が小さいので入れ替えは行わない。
  - これで全ての比較（並べ替え）が完了し，[3,2,4] が [2,3,4] と昇順に並べ替えられる。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Data = [64, 34, 25, 12, 22, 11, 90]
n = len(Data)

for i in range(n):
  for j in range(n - i - 1):
    if Data[j] > Data[j + 1]:
      # 要素の交換
      temp = Data[j]
      [          ]
      Data[j + 1] = temp

print(' 並べ替え後 :', Data)
```

① `Data[j] > Data[i]`

② `Data[j - 1] = Data[j]`

③ `Data[j] = Data[j + 1]`

④ `Data[i] = Data[i + 1]`

＜MEMO＞

# 上級編　第 44 講　確認問題

以下の条件で，選択法（選択ソート）を使って配列の要素を昇順に並べ替えるプログラムを作成する。

・選択法とは，並べ替えたいデータの中から最小値（または最大値）を見つけ，それを左から順に入れていくことによって昇順（または降順）に並べ替えを行う整列アルゴリズムである。

・今回は，昇順に並べ替えを行う。

・例えば，[3,4,2] というデータがある場合，次のような手順で並べ替えを行う。

　◦はじめに,配列全体から最小の値である２を選ぶ。

　◦この２と先頭の３を入れ替えることで，[2,4,3] となる。これで一番小さい値の２が最初の位置に確定する。

　◦次に，残りの部分配列 [4,3] から最小の値である３を選び，これを先頭の４と入れ替えることで，[2,3,4] となる。

　◦これで全ての比較（並べ替え）が完了し，[3,4,2] が [2,3,4] と昇順に並べ替えられる。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Data = [64, 34, 25, 12, 22, 11, 90]
n = len(Data)

for i in range(n - 1):
  min_index = i
  for j in range(i + 1, n):
    if [          ]:
      min_index = j

  # 交換処理
  temp = Data[i]
  Data[i] = Data[min_index]
  Data[min_index] = temp

print(' 並べ替え後 :',Data)
```

① Data[j] < Data[min_index]

② Data[j] > Data[min_index]

③ Data[j] < Data[i]

④ Data[j] != Data[min_index]

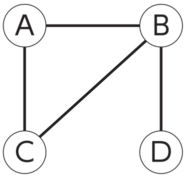
上級編

# 第45講 グラフ理論

## 例題

グラフ理論とは，物事の間の関係を数学的に表現し，分析するための理論である。
このグラフ理論を用いた簡単な例として，友人関係を調べるプログラムを以下の条件で作成する。

・A，B，C，D の 4 人の関係を表したグラフは次のようになる。




・線でつながっている人が友人関係となる。
・例えば，A は B と C と友達であるが，D とは友達ではない。
・友人関係ではない場合を 0，友人関係である場合を 1 とする表を作成すると次のようになる。

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 |
| C | 1 | 1 | 0 | 0 |
| D | 0 | 1 | 0 | 0 |

・これを 2 次元配列（配列名：Data）とした場合，2 行 3 列目のデータを取得する場合は添字は 0 から始まるので，「Data[1][2]」というように記述する。
・name_index ＝ Human.index(name) という命令は，配列 Human の中で変数 name の値と一致する要素の添字（インデックス）を取得し，変数 name_index に代入するものである。（例：Human=['A','B','C','D']，name='A' のとき，Human.index(name) は 0）

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Data = [
#  A  B  C  D
  [0, 1, 1, 0], # A
  [1, 0, 1, 1], # B
  [1, 1, 0, 0], # C
  [0, 1, 0, 0]  # D
]
Human = ['A', 'B', 'C', 'D']
Friends = []
name = input('A から D の中で，友人関係を知りたい人を入力してください。( 例 :B) : ')
name_index = Human.index(name) # 入力された人に対応する添字を取得

for i in range(len(Data)):
  if ____________:
    Friends.append(Human[i])

# 結果表示 ( カンマ区切りで表示 )
print(name, ' の友達は ', Friends)
```

① Data[name_index][i] == 0

② Data[name_index][i] == 1

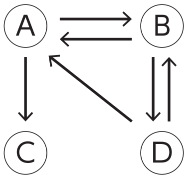
③ Data[name_index][i] != 1

④ Data[name_index] == 0

# 上級編　第 45 講　確認問題

グラフ理論の「有向グラフ」とは，関係を示す線に矢印がついている関係図のことで，物事の関係性を把握するのに適している。
この有向グラフの考え方を用いて，SNS のフォロー関係を調べるプログラムを以下の条件で作成する。

・A，B，C，D の４人のフォロー関係を表したグラフは次のようになる。




・グラフ内で，矢印が指し示している先がフォローしている人となる。
・例えば, D は A と B をフォローしていて, B からフォローされている。
・フォローしていない場合を 0，フォローしている場合を 1 とする表を作成すると次のようになる。

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 |
| B | 1 | 0 | 0 | 1 |
| C | 0 | 0 | 0 | 0 |
| D | 1 | 1 | 0 | 0 |

・これを２次元配列（配列名：Data）とした場合，２行３列目のデータを取得する場合は添字は０から始まるので,「Data[1][2]」というように記述する。
・name_index = Human.index(name) という命令は，配列 Human の中で変数 name の値と一致する要素の添字を取得し，変数 name_index に代入するものである。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Data = [
  [0, 1, 1, 0], # A
  [1, 0, 0, 1], # B
  [0, 0, 0, 0], # C
  [1, 1, 0, 0]  # D
]

Human = ['A', 'B', 'C', 'D']
ToFollow = [] # フォローしている人を格納
FromFollow = [] # フォローされている人（フォロワー）を格納

name = input('AからDの中で，フォロー関係を知りたい人を入力してください。（例:B）: ')
name_index = Human.index(name) # 入力された人に対応する添字を取得

for i in range(len(Data)):
  if Data[name_index][i] == 1:
    ToFollow.append(Human[i])
  if [          ]:
    FromFollow.append(Human[i])

# 結果表示
print(name, 'がフォローしている人は')
print(ToFollow)
print(name, 'をフォローしている人は')
print(FromFollow)
```

① `Data[name_index][i] == 0`

② `Data[i][name_index] != 1`

③ `Data[i][name_index] == 1`

④ `FromFollow[i] == name`

上級編

# 第 46 講　待ち行列

## 例題

人気ラーメン店の行列シミュレーションプログラムを，以下の条件で作成する。

・ラーメン店の座席数，調理時間やお客さんの来るタイミングなどは以下の通りである。

　◦ラーメン店の座席数：８席

　◦ １杯のラーメンを作るのにかかる時間：３分

　◦ １杯のラーメンを食べるのにかかる時間：10 分

　◦したがって，一人の客が座席についてから店を出るまでの合計時間：13 分（調理時間３分＋食事時間 10 分）

　◦お客さんは１分間に０人～２人到着する。

　◦座席が満席の場合は，行列に並ぶ。

　◦座席が空いてから片づけを行い，次の客が座席に座るまで１分を要する。

・以下のような形式で，１分ごとに到着客，行列客，席に座っている客の人数を出力する。

実行結果

```
1 分：到着客：1　 行列客： 0　 席に座っている客： 1
2 分：到着客：1　 行列客： 0　 席に座っている客： 2
3 分：到着客：2　 行列客： 0　 席に座っている客： 4
4 分：到着客：0　 行列客： 0　 席に座っている客： 4
5 分：到着客：0　 行列客： 0　 席に座っている客： 4
6 分：到着客：2　 行列客： 0　 席に座っている客： 6
7 分：到着客：2　 行列客： 0　 席に座っている客： 8
8 分：到着客：0　 行列客： 0　 席に座っている客： 8
9 分：到着客：2　 行列客： 2　 席に座っている客： 8
10 分：到着客：2　 行列客： 4　 席に座っている客： 8
```

・seats.count(0) という命令は，配列 seats 内の要素「0」の個数を返すものである。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

seats = [0, 0, 0, 0, 0, 0, 0, 0] # 座席配列 ( 店を出るまでの残り時間を格納 )
seats_num = len(seats) # 座席の数
cook_time = 3 # 調理時間
eat_time = 10 # 食べる時間
total_time = cook_time + eat_time # 調理時間 + 食べる時間
waiting = 0 # 行列人数
# 30 分間のシミュレーション
for minute in range(30):
  arrival = random.randint(0, 2)
  waiting = waiting + arrival
  # 各席の状態更新
  for i in range(seats_num):
    if seats[i] > 0:
      seats[i] = □
    elif waiting > 0:
      seats[i] = total_time
      waiting = waiting - 1
  print(minute + 1, '分：到着客:', arrival, '　行列客：', waiting, '　席に座っている客：', seats_num - seats.count(0))
```

① seats[i] + 1

② seats[i] - 1

③ seats[i + 1]

④ seats[i] + 1

# 上級編　第 46 講　確認問題

以下の条件で，一方通行の道路における交通渋滞シミュレーションプログラムを作成する。

・この一方通行の道路では，渋滞が頻繁に発生する。

・この問題に対処するために，横断歩道の信号のタイミングを調整することで渋滞を緩和できるかシミュレーションすることにした。

・道路の現状と横断歩道信号のタイミングに関する調査から，次の情報が得られた。

　◦青信号の間，10 秒間に最大 10 台の車が道路を通過できる。

　◦信号は 60 秒間の青信号と 30 秒間の赤信号が交互に変わる。

　◦ 10 秒間に道路に到着する車の台数は，ランダムに 5 台から 10 台の間で変動する。

・max 関数は，引数の中で一番大きいものを返す関数である。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

totaltime = 18    # 観測時間：単位 10 秒（180 秒なら 18）
passtime = 10     # 通過可能台数
signal_b = 6      # 青信号の時間：単位 10 秒
signal_r = 3      # 赤信号の時間：単位 10 秒
blue_flag = True  # 青信号の場合「True」
arrive = 0        # 車の到着台数
wait = 0          # 渋滞台数

print('秒　到着台数　渋滞台数')

for t in range(1, totaltime + 1):
```

```
    arrive = random.randint(5, 10) # 10 秒間に 5 ～ 10 台ランダムに車が到着する
    mod = (t - 1) % (signal_b + signal_r) + 1
    blue_flag = (mod <= signal_b)
    if blue_flag: # 青信号ならば
        wait = max([    ], 0)
    else: # 赤信号ならば
        wait = wait + arrive # 通過できない車はすべて渋滞に加算される
    print(t * 10, '¥t', arrive, '¥t', wait, '¥t')
```

実行結果

```
秒   到着台数  渋滞台数
10        5        0
20        9        0
30        9        0
40        8        0
50        9        0
60       10        0
70        6        6
80        9       15
90        8       23
100       6       19
110      10       19
120       8       17
130       6       13
140       8       11
150       6        7
160      10       17
170       5       22
180       7       29
```

① `wait + arrive - passtime`

② `wait + arrive`

③ `wait + arrive + passtime`

④ `arrive - passtime`

上級編

# 第47講 パリティチェック

## 例題

以下の条件で，偶数パリティ方式を用いた，誤り検出を行うプログラムを作成する。

・偶数パリティとは，誤り検出方式の一つで，ビット列中に含まれる「1」の個数が奇数個ならパリティビットを「1」に，偶数個なら「0」にする方式である。

・例えば，「1010」というビット列は 1 の数が偶数個なので，末尾にパリティビットを加えると「10100」となる。

・一方，「1000」というビット列は 1 の数が奇数個なので，末尾にパリティビットを加えると「10001」となる。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
count = 0
bit_retsu = input(' パリティビットを含めたビット列を入力してください： ')

for i in range(len(bit_retsu) - 1):
  if bit_retsu[i] == '1':
    count = count + 1

if [          ]:
  parity_bit = '0'
else:
  parity_bit = '1'
# パリティビットの検証
if parity_bit == bit_retsu[-1]: # bit_retsu[-1] は，bit_retsu 配列の最後の要素を取得
  print(' ビット列は正しいです。パリティビット :', parity_bit)
else:
  print(' ビット列は誤っています。正しいパリティビットは ', parity_bit , ' です。')
```

① count % 2 == 0

② count % 2 == 1

③ count == 0

④ count / 2 == 0

＜MEMO＞

# 上級編　第 47 講　確認問題

以下の条件で，JAN コードのチェックディジットを求め，誤りチェックを行うプログラムを作成する。

・JAN コードとは，バーコードの規格の一つであり，どのメーカーのどの商品であるかを示す国際標準の商品識別コードである。

・13 桁の JAN コードの場合，末尾 1 桁がチェックディジットである。

1 234567 89012⑧

チェックディジット

・チェックディジットとは，入力の誤りを検出したり，捏造を防止したりするために，一定のルールに従って元の符号に加えられる数値や記号のことである。

・JAN コードのチェックディジットは，次のような手順で算出する。

⑴　桁数が奇数桁のコードの和を求める（チェックディジットを除く）。

⑵　桁数が偶数桁のコードの和を求め，それを 3 倍する。

⑶　⑴の結果と⑵の結果の和を求める。

⑷　⑶の結果の 1 の位の数を 10 から引いた結果の 1 桁目がチェックディジットである。　例：1 の位が 3 のとき　10-3=7

・入力が 13 桁でなかった場合は exit() という命令を用いて，その時点でプログラムを終了する。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
Jan_code = input('13 桁の JAN コードを入力してください：')

if len(Jan_code) != 13:
  print('入力されたコードは 13 桁ではありません。')
  exit()

odd_sum = 0 # 奇数桁の合計値を格納
even_sum = 0 # 偶数桁の合計値を格納

for i in range(12):
  if i % 2 == 0:
    odd_sum = odd_sum + int(Jan_code[i])
  else:
    even_sum = even_sum + int(Jan_code[i])

total = [          ]

ones_place = total % 10

if ones_place == 0:
  check_digit = 0
else:
  check_digit = 10 - ones_place

# チェックディジットの検証
if check_digit == int(Jan_code[-1]):
  print('JAN コードは正しいです。チェックディジット：', check_digit)
else:
  print('JAN コードは誤っています。正しいチェックディジットは', check_digit, 'です。')
```

① odd_sum * 3 + even_sum

② odd_sum + even_sum * 3

③ odd_sum * even_sum

④ odd_sum + 3 + even_sum

上級編

# 第 48 講 すごろくゲーム

## 例題

1 人用の簡易的なすごろくゲームのプログラムを，以下の条件で作成した。

- プレイヤーは 1 人。
- プレイヤーは，六面のサイコロを順番に振って出た目の数だけ進む。
- 途中，特定のマス（10 マス目）に罠があり，そこに止まると 5 マス戻る。
- スタートを 0 マス目として，20 マス目以上に達したらゴールとする。
- ゴールまでに振ったサイコロの回数を，出力する。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random
# 初期化
player_position = 0 # プレイヤーの位置
trap_position = 10 # 10 マス目に罠を設置
goal_position = 20
roll_count = 0 # サイコロを振った回数

while player_position < goal_position:
  input(' サイコロを振ってください (Enter を押す ) ！ ')
  roll = random.randint(1, 6)
  [                    ]
  print(' サイコロの出目：', roll, ' ', player_position, ' マス目に移動しました。')
  roll_count = roll_count + 1

  if player_position == trap_position:
    print(' 移動先に罠がありました！　ここから 5 マス戻ります！ ')
    player_position = player_position - 5

  if player_position >= goal_position:
    print(' ゴールに到達しました！おめでとうございます！ ')
    print(' 合計でサイコロを ', roll_count, ' 回振りました。')

print(' ゲーム終了！ ')
```

① `player_position = roll`

② `player_position = goal_position + roll`

③ `trap_position == player_position + roll`

④ `player_position = player_position + roll`

# 上級編　第 48 講　確認問題

1 人用の簡易的なすごろくゲームのプログラムを，以下の条件で作成した。

- プレイヤーは 1 人。
- プレイヤーは，六面のサイコロを順番に振って出た目の数だけ進む。
- 毎回 15,16,17 マス目のどこか 1 箇所にお化けがランダムに出現し，そこに止まると 5 マス戻る。
- スタートを 0 マス目として，20 マス目以上に達したらゴールとする。
- ゴールまでに振ったサイコロの回数を出力する。

以下のプログラムの空欄に入る最も適切なものを，あとの①～④のうちから一つ選びなさい。

```
import random

# ゲームの基本設定
goalposition = 20
Ghostpositions = [15, 16, 17]  # お化けがいる可能性のある位置
player_position = 0  # プレイヤーの初期位置
roll_count = 0  # サイコロを振った回数

# ゲーム開始
while player_position < goalposition:
  input('サイコロを振ってください(Enterを押す)！')
  roll = random.randint(1, 6)
  player_position = player_position + roll
  roll_count = roll_count + 1
  print('サイコロの出目：', roll, ' ', player_position, 'マス目に移動しました。')

  # お化けのチェック
  ghost_position = random.choice(Ghostpositions)
  if [                    ]:
    print('お化けに遭遇！ 5マス戻ります。')
    player_position = player_position - 5
```

```

# ゴールに到達
print('ゴールに到達しました！おめでとうございます！')
print('合計でサイコロを', roll_count, '回振りました。')
```

実行結果

```
サイコロを振ってください(Enter を押す)！
サイコロの出目：6　6 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目：3　9 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目：2　11 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目：5　16 マス目に移動しました。
お化けに遭遇！5 マス戻ります。
サイコロを振ってください(Enter を押す)！
サイコロの出目：5　16 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目：3　19 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目：5　24 マス目に移動しました。
ゴールに到達しました！おめでとうございます！
合計でサイコロを 7 回振りました。
```

① `player_position < ghost_position`

② `player_position > ghost_position`

③ `player_position == ghost_position`

④ `player_position != ghost_position`

# 例題解答

# 中級編　例題　解答

## 第15講　関数

例題１　解答：②

例題２　解答：②

## 第16講　配列と乱数

例題　解答：③

## 第17講　２次元配列

例題　解答：②

## 第18講　input と random.choice

例題　解答：①

## 第19講　引数が配列の関数

例題　解答：①

## 第20講　１から n までの和

例題　解答：②

## 第21講　○以上○以下の判定

例題　解答：①

## 第22講　２つの数字の並べ替え

例題　解答：②

## 第23講　合計と平均

例題　解答：①

## 第24講　for 文の入れ子

例題　解答：④

## 第25講　for 文と if 文の入れ子

例題　解答：③

## 第26講　最大値と最小値

例題　解答：②

## 第27講　カウントダウン

例題　解答：③

## 第28講　while 文と if 文の入れ子

例題　解答：③

## 第29講　10 進法から 2 進法への変換

例題　解答：④

## 第30講　ソート（並べ替え）

例題　解答：④

## 第31講　配列への要素の追加

例題　解答：③

## 第32講　2 次元配列の応用

例題　解答：②

## 第33講　辞書型

例題　解答：②

# 上級編　例題　解答

## 第34講　コインの枚数

例題　解答：④

## 第35講　駐車料金の判定

例題　解答：④

## 第36講　割引のある料金

例題　解答：④

## 第37講　テストの集計

例題　解答：①

## 第38講　じゃんけん

例題　解答：①

## 第39講　サイコロの出目

例題　解答：③

## 第40講　中央値の判定

例題　解答：①

## 第41講　素数判定

例題　解答：③

## 第42講　素因数分解

例題　解答：②

## 第43講　倍数の判定・FizzBuzz

例題　解答：④

## 第44講　バブルソート・選択ソート

例題　解答：③

## 第45講　グラフ理論

例題　解答：②

## 第46講　待ち行列

例題　解答：②

## 第47講　パリティチェック

例題　解答：①

## 第48講　すごろくゲーム

例題　解答：④

# 確認問題解答

# 初級編　第 1 講　確認問題　解答

正解：③

解説：

print 関数は，( ) 内に変数を記述することで，その変数の値を表示することができる。

```
name = 'Tokyo'
print(name)
```

実行結果

```
Tokyo
```

# 初級編　第２講　確認問題　解答

正解：②

解説：

print 関数の引数内の `ringo * 3 + mikan * 2` では，りんご３個とみかん２個の合計金額を求めている。その金額から，割引クーポンの代金を引けばよい。よって，空欄には引き算を表す，- が入る。

```
# 支払額
ringo = 200
mikan = 100
waribiki = 50
print(ringo * 3 + mikan * 2 - waribiki, '円の支払い')
```

実行結果

```
750 円の支払い
```

# 初級編　第３講　確認問題　解答

正解：④

解説：

print 関数では，( ) 内にカンマ区切りで値や引数を記述すると，それらはスペース（空白）で区切られ，並んで出力される。また，商を求めるときは //，余りを求めるときは，% の演算子を用いる。よって，正しい組み合わせは④である。

```
wararerukazu = 25
warukazu = 7
q = wararerukazu // warukazu
r = wararerukazu % warukazu
print('商', q, '余り', r)
```

実行結果

```
商 3 余り 4
```

# 初級編　第 4 講　確認問題　解答

正解：①

解説：

!= は，両辺の値が等しくないときに，True を返す演算子である。ここでは，a と b の値は等しくないので，True が結果となる。

```
a = 7
b = 2

print(a != b)
```

実行結果

```
True
```

# 初級編　第５講　確認問題　解答

正解：③

解説：

Python では，配列の添字は 0 から始まる。よって，Numbers[2] とは，配列のデータの左から３番目の値を指定している。

```
Numbers = [1, 2, 3, 4, 5]
print(Numbers[2])
```

実行結果

```
3
```

# 初級編　第6講　確認問題　解答

正解：①

解説：

random の randint 関数は，整数の乱数を生成する関数である。ここでは，サイコロの６面の１～６の整数を生成するため，randint 関数が適切である。

```
import random

dice_roll = random.randint(1, 6)
print(dice_roll)
```

実行結果

```
2
```

# 初級編　第7講　確認問題　解答

正解：②

解説：

len 関数の引数に配列を指定することで，その配列の要素数を取得できる。配列 Animals には 'cat'，'dog'，'fish'，'bird' の４つのデータが格納されているため，結果は４となる。

```
Animals = ['cat', 'dog', 'fish', 'bird']
print(len(Animals))
```

実行結果

```
4
```

# 初級編　第 8 講　確認問題　解答

正解：④

解説：

range 関数の引数が 3 個ある場合は，開始値，終了値 -1，間隔値をそれぞれ指定している。問題文の [1, 4, 7, 10, 13, 16, 19] からは，1 が開始値，19 が終了値 -1 の値，間隔値が 3 であることが読み取れる。よって，④が正解である。

```
print(list(range(1, 20, 3)))
```

実行結果

```
[1, 4, 7, 10, 13, 16, 19]
```

# 初級編　第9講　確認問題　解答

正解：②

解説：

問題文のプログラムには，in の後に配列 [2, 4, 6, 8] が指定されているため，実行結果は，2,4,6,8 が表示される。ここから，開始値 2，終了値 8，間隔値 2 であることがわかる。

このプログラムと同じ実行結果を表示させるためには，range 関数の 1 つ目の引数に開始値として 2，2 つ目の引数に終了値 +1 の値として 9，3 つ目の引数に間隔値として 2 を指定すればよい。よって，正解は②となる。

```
for i in range(2, 9, 2):
  print(i)
```

実行結果

```
2
4
6
8
```

# 初級編　第 10 講　確認問題　解答

正解：③

解説：

x には 3 が代入されているので，if 条件式の判定は False となる。よって，if ブロックの処理は実行されず，何も表示されない。

```
x = 3
if x > 10:
  print(x, 'は 10 より大きい')
```

# 初級編　第 11 講　確認問題　解答

正解：①

解説：

x には 10 が代入されているので，if 条件式の値は False となる。よって，else ブロックの処理が実行される。

```
x = 10
if x < 10:
  print(x, 'は10より小さい')
else:
  print(x, 'は10以上')
```

実行結果

```
10 は10以上
```

# 初級編　第 12 講　確認問題　解答

正解：④

解説：

if-elif-else 文では，上から順番に条件を評価していき，True となった条件のインデントされたブロックの処理を行う。ここでは，x には 10 が代入されているため，elif 条件式に当てはまり，elif ブロック内の処理が実行される。

```
x = 10
if x < 10:
  print(x, 'は 10 より小さい')
elif x == 10:
  print(x, 'は 10 に等しい')
else:
  print(x, 'は 10 より大きい')
```

実行結果

```
10 は 10 に等しい
```

# 初級編　第 13 講　確認問題　解答

正解：②

解説：

while 文では，条件式が True（ここでは，count <= 3）であるとき，ブロック内の処理を続ける。下記の表では，while 文のループ中の処理をまとめている。

| n 回目 | 条件判定時の count の値 | 条件式の結果 | 結果 |
|---|---|---|---|
| 1 回目 | 1 | True | Hello が出力される。その後，count の値が 1 加算される |
| 2 回目 | 2 | True | Hello が出力される。その後，count の値が 1 加算される |
| 3 回目 | 3 | True | Hello が出力される。その後，count の値が 1 加算される |
| 4 回目 | 4 | False | ループが終了する |

よって,「Hello」は 3 回出力されるため, 正解は②である。

```
count = 1
while count <= 3:
  print('Hello')
  count = count + 1
```

実行結果

```
Hello
Hello
Hello
```

# 初級編　第 14 講　確認問題　解答

正解：①

解説：

a と b の値を or 演算子で演算すると，両方の値が True であるため，result には True の値が返される。

```
a = True
b = True
result = a or b
print(result)
```

実行結果

```
True
```

# 中級編　第 15 講　確認問題　解答

正解：②

解説：

multiply 関数は，２つの引数を掛けた結果を戻り値とする関数である。よって，３と５の積である 15 が実行結果として表示される。

```
def multiply(a, b):
  return a * b

multiply(3, 5)
```

実行結果

```
15
```

# 中級編　第 16 講　確認問題　解答

正解：④

解説：

4 行目の `number = random.randint(0,3)` では，ランダムに生成された 0 から 3 までの整数の値が，変数 number に代入される。

そのため変数 number を配列 Omikuji の添字に指定することで，おみくじの結果をランダムに出力させることが可能である。よって正解は，④ `Omikuji[number]` である。

```
01 import random
02 
03 Omikuji = ['大吉', '中吉', '小吉', '凶']
04 number = random.randint(0,3)
05 print(Omikuji[number])
```

実行結果

```
小吉
```

# 中級編　第 17 講　確認問題　解答

正解：③

解説：

問題文の 2 次元配列では，1 つ目の添字が何番目の科目であるかを指定し，2 つ目の添字がその科目の誰の点数であるかを指定している。下記は，2 次元配列の内容を表にまとめたものである。

| Alice | Bob | Charie |
|---|---|---|
| 85 | 92 | 100 |
| 90 | 88 | 95 |
| 78 | 81 | 76 |

ここでは，Alice の 3 つ目の科目の点数である「78」が表示されたので，1 つ目の添字は 3 つ目の科目を指定する「3」を，2 つ目の添字は Alice の得点を指定する「0」であることがわかる。よって，正解は③ Students_and_scores[3][0] である。

```
Students_and_scores = [
                         ['Alice', 'Bob', 'Charlie'],
                         [85, 92, 100],
                         [90, 88, 95],
                         [78, 81, 76]
                        ]
value = Students_and_scores[3][0]
print(value)
```

実行結果

```
78
```

# 中級編　第 18 講　確認問題　解答

正解：②

解説：

6 行目では，random.choice 関数の引数に指定した配列の要素をランダムに 1 つ抽出し，その値を変数 computuer_obj に代入している。

このプログラムでは，コンピュータに「〇」「△」「×」のいずれかの値をランダムに出力させるため，その 3 つの要素を持つ配列 Obj を random.choice 関数の引数として指定すればよい。よって，正解は② Obj である。

```
01 import random
02
03 user_obj = input('〇，△，×の中から好きな形を入力して：')
04
05 Obj = ['〇', '△', '×']
06 computer_obj = random.choice(Obj)
07
08 print('あなたは', user_obj, 'を選びました。')
09 print('コンピュータは', computer_obj, 'を選びました。')
```

実行結果

```
〇，△，×の中らか好きな形を入力して：△
あなたは △ を選びました。
コンピューターは 〇 を選びました。
```

# 中級編　第 19 講　確認問題　解答

正解：①

解説：

calculate_sum 関数は，配列 Arr の要素を 1 つずつ取り出し，その合計値を関数の戻り値として返している。

そのため，配列 Numbers の要素の合計を表示するためには，caluculate_sum 関数を呼び出し，その引数に配列 Numbers を指定すればよい。よって，正解は③ `calculate_sum(Numbers)` である。

```
def calculate_sum(Arr):
  total = 0
  for num in Arr:
    total = total + num
  return total

Numbers = [12, 14, 15, 21, 25]
result = calculate_sum(Numbers)
print(result)
```

実行結果

```
87
```

# 中級編　第 20 講　確認問題　解答

正解：④

解説：

calculate_sum 関数は，1 から n までの合計値を求める関数である。3 行目の for 文では，1 から n までの間の整数を 1 つずつ取り出して変数 i に代入している。

そのため 4 行目では，合計値を表す変数 total に取り出した i の値を加算する処理を行う必要がある。よって，正解は④の `total + i` である。

```
01 def calculate_sum(n):
02   total = 0
03   for i in range(1, n + 1):
04     total = total + i
05   return total
06 
07 n = 5
08 result = calculate_sum(n)
09 print(result)
```

実行結果

```
15
```

# 中級編　第 21 講　確認問題　解答

正解：①

解説：

if-elif-else 文では，上から順番に条件を評価していき，最初に True となった条件式内の処理を行う。

問題文から，A と表示されるのは，得点が 60 以上の場合である。よって，[ア]には〇〇以上を表す `score >= 60` が当てはまる。

また，B と表示されるのは，得点が 50 以上 60 未満の場合である。よって，[イ]には〇〇以上を表す `score >= 50` が当てはまる。

```
def check_grade(score):
  if score >= 60:
    return 'A'
  elif score >= 50:
    return 'B'
  else:
    return 'C'

score = 55
print(check_grade(score))
```

実行結果

```
B
```

# 中級編　第 22 講　確認問題　解答

正解：①

解説：

sort_two_numbers 関数では，2 つの数 a，b を大きい順に並び替える。そのため，2 つの数が大きい順に並んでいない（a のほうが b より小さい）ときに順番を入れ替える処理を行えばよい。よって，if 文の条件式には①の `a < b` が当てはまる。

```
def sort_two_numbers(a, b):
  if a < b:
    temp = a
    a = b
    b = temp
  return a, b

result = sort_two_numbers(2, 5)
print(result)
```

実行結果

```
(5, 2)
```

# 中級編　第 23 講　確認問題　解答

**正解：①**

**解説：**

4 行目の `for num in Data` では，配列 Data の要素を取り出し，変数 num に代入するという処理を，配列 Data の要素ぶん繰り返している。

よって，配列 Data の要素の合計を求めるためには，合計値を表す変数 total に変数 num を加算する `total + num` が空欄に入ればよい。

```
01 Data = [10, 20, 30, 40, 50]
02 total = 0
03 
04 for num in Data:
05   total = total + num
06 
07 print(total)
```

実行結果

```
150
```

# 中級編　第 24 講　確認問題　解答

正解：③

解説：

このプログラムでは，１～９の段までの九九の結果を表示するため，かけられる数（変数 i）もかける数（変数 j）も１から９までの数をもつ。よって，for 文で指定する範囲は，いずれも range(1, 10) となる。

なお，range 関数は，１つ目の引数には開始値を，２つ目の引数には終了値 +1 の値を指定することに注意する。

```
for i in range(1, 10):
  for j in range(1, 10):
    value = i * j
    print(i, '×', j, '=', value)
```

実行結果

```
1 × 1 = 1
1 × 2 = 2
...
9 × 8 = 72
9 × 9 = 81
```

# 中級編　第 25 講　確認問題　解答

正解：①

解説：

３行目の `for i in range(1, n + 1)` は，1 ～ n の整数を順番に取り出す処理である。そして，４行目の `if i % 2 == 1` は，i の値を２で割って余りが 1 であるかどうか，つまり i の値が奇数がどうかを判定する条件式である。

よって，この if 文の条件式が True であるときに，合計値を表す変数 total に i の値を加算すればよいので，６行目は `total = total + i` となる。

```
01 def calculate_odd_sum(n):
02   total = 0
03   for i in range(1, n + 1):
04     if i % 2 == 1:
05       total = total + i
06   return total
07
08 n = 5
09 result = calculate_odd_sum(n)
10 print(result)
```

実行結果

```
9
```

# 中級編　第 26 講　確認問題　解答

正解：①

解説：

ここでは２行目で，配列 Numbers[0] の値を最大値と仮定して変数 max_value に代入している。３行目の for 文では，配列 Numbers の要素を変数 num に順番に代入している。

そのため４行目の if 文では，最大値と仮定した変数 max_value と num の値の大小を比較して，num のほうが大きい場合は，変数 max_value に num の値を代入する必要がある。よって，if 文の条件式には①の `num > max_value` が当てはまる。

```
01 def find_max(Numbers):
02   max_value = Numbers[0]
03   for num in Numbers:
04     if num > max_value:
05       max_value = num
06 
07   return max_value
08 
09 Numbers = [90, 80, 55, 77, 92]
10 max_value = find_max(Numbers)
11 print(max_value)
```

実行結果

```
92
```

# 中級編　第 27 講　確認問題　解答

正解：②

解説：

実行結果には，5 から 1 までの値が 1 ずつ減って表示されている。つまり，while 文の条件式が True である間（`while count > 0`），変数 count の値を 1 ずつ減らす処理を空欄に当てはめればよい。よって，②の `count - 1` が正解である。

```
count = 5
while count > 0:
  print(count)
  count = count - 1
```

実行結果

```
5
4
3
2
1
```

# 中級編　第 28 講　確認問題　解答

正解：①

解説：

2 行目 while 文の条件式が True であるとき（変数 x の値が 5 より小さいとき），次の行の if 条件式の判定に移る。

3 行目 if 文の条件式では，x の値が 3 であるときは，if 文の処理が実行され，Three と表示される。x の値が 3 以外のときは，else 文の処理に入り，x の値を表示する。

x の値は初期値 0 から 5 になるまで 1 ずつ加算されるため，`0 1 2 Three 4` と表示される。なお，x の値が 5 になったときに while 文の条件式が False となるため，5 は表示されない。

```
01 x = 0
02 while x < 5:
03   if x == 3:
04     print('Three')
05   else:
06     print(x)
07   x = x + 1
```

実行結果

```
0
1
2
Three
4
```

# 中級編　第 29 講　確認問題　解答

正解：②

解説：

10 進法から 2 進法に変換する場合は，10 進数の整数を 2 で割り，その商が 0 になるまでの余りを並べればよい。3 ～ 5 行目の繰り返し処理では，10 進数の値が 0 より大きい場合，その整数を 2 で割ったときの余りを変数 binary が文字列として保有している。

10 進法の整数の商が 0 になるまで 2 で割る処理を続けるため，6 行目には，10 進法の整数 num を 2 で割る `num = num // 2` が当てはまる。

```
01 num = 26
02 binary = ''
03 while num > 0:
04   remainder = num % 2
05   binary = str(remainder) + binary
06   num = num // 2
07
08 print(binary)
```

実行結果

```
11010
```

# 中級編　第 30 講　確認問題　解答

正解：④

解説：

配列の要素を大きい順に並び替えるには，配列の要素を隣同士で比較して，前の値のほうが小さい場合は，要素の順番を入れ変える処理を行えばよい。

また，5 行目 ~7 行目の処理では，Numbers[j + 1] をNumbers[j] の位置に入れ替える処理をしている。これらのことから，Numbers[j] のほうが，次の要素の値Numbers[j + 1] より小さいときに，if 文の処理を行う条件式を指定すればよい。 よって，正解は④の Numbers [j] < Numbers [j + 1] となる。

```
01 Numbers = [3, 1, 4, 1, 5]
02 for i in range(len(Numbers)):
03   for j in range(len(Numbers) - i - 1):
04     if Numbers[j] < Numbers[j + 1]:
05       temp = Numbers[j]
06       Numbers[j] = Numbers[j + 1]
07       Numbers[j + 1] = temp
08 print(Numbers)
```

実行結果

```
[5, 4, 3, 1, 1]
```

# 中級編　第 31 講　確認問題　解答

正解：③

解説：

配列の末尾に要素を追加する場合，append() メソッドを使用する。よって，正解は Subjects.append(' 英語 ') の③である。

```
Subjects = [' 国語 ', ' 数学 ']
Subjects.append(' 英語 ')
print(Subjects)
```

実行結果

```
[' 国語 ', ' 数学 ', ' 英語 ']
```

# 中級編　第 32 講　確認問題　解答

正解：③

解説：

実行結果では，この２次元配列の要素が全て出力されている。

全ての要素を取り出すためには，この２次元配列の要素の数ぶん，繰り返し処理を行えばよい。そのため，len 関数で配列の要素数を取得し，その値を num の値に代入すればよい。よって５行目には，`num = len(Fruits_and_numbers[1])` が当てはまる。

```
01 Fruits_and_numbers = [
02                       ['りんご', 'みかん', 'バナナ'],
03                       [10, 12, 18]
04                      ]
05 num = len(Fruits_and_numbers[1])
06 for i in range(num):
07   print(Fruits_and_numbers[1][i], '個の', Fruits_and_numbers[0][i])
```

実行結果

```
10 個の りんご
12 個の みかん
18 個の バナナ
```

# 中級編　第 33 講　確認問題　解答

正解：②

解説：

辞書型の値を呼び出す際は，その値と対応するキーを [] に指定する。

7 行目では，辞書型 fruits のキー 'apple' と 'cherry' を指定してそのキーと対応する値を呼び出し，2 つの値を加算している。よって 100 と 300 が呼び出され，その合計値 400 が表示される。

```
01 fruits = {
02   'apple': 100,
03   'banana': 200,
04   'cherry': 300
05 }
06
07 total = fruits['apple'] + fruits['cherry']
08 print(total)
```

実行結果

```
400
```

# 上級編　第 34 講　確認問題　解答

正解：④

解説：

7 行目の Kosuu[i] = kingaku // Okashi[i] では，配列 Okashi の i 番目のお菓子の購入個数を配列 Kosuu[i] に代入している。そのため，続く 8 行目では，配列 Okashi の i 番目のお菓子を購入したあとの残金を，変数 kingaku に代入すればよいと考えられる。残金は，現在の金額から i 番目のお菓子の金額を購入した余りの金額であるため，空欄には，kingaku % Okashi[i] が当てはまる。よって，正解は④ kingaku % Okashi[i] である

```
01 Okashi = [350, 170, 90, 55, 30, 20] # 売っているお菓子の金額（値段の降順）
02 Kosuu = [0, 0, 0, 0, 0, 0]
03 
04 kingaku = int(input('予算を入力してください：'))
05 
06 for i in range(len(Okashi)):
07   Kosuu[i] =  kingaku // Okashi[i]
08   kingaku = kingaku % Okashi[i]
09 
10 for i in range(len(Okashi)):
11   print(Okashi[i],'円のお菓子：',Kosuu[i],'個')
```

実行結果

```
予算を入力してください：3334
350 円のお菓子： 9 個
170 円のお菓子： 1 個
90 円のお菓子： 0 個
55 円のお菓子： 0 個
30 円のお菓子： 0 個
20 円のお菓子： 0 個
```

# 上級編　第 35 講　確認問題　解答

正解：③

解説：

乗車料金を求めるためには，入力した乗車距離が初乗り距離より大きいか，それ以下かどうかを判断する必要がある。

8 行目の if 文では，乗車距離が初乗り距離より大きい場合の条件を指定している。したがって，この条件式が True であるときは，「(乗車距離 - 初乗り距離) × 1km ごとの料金 + 初乗り料金」の計算結果が支払い料金になると考えられる。

よって，正解は③ (jousya_kyori - hatsunori_kyori) * kiro_ryoukin + hatsunori_ryoukin である。

```
01 hatsunori_ryoukin = 500 # 初乗り料金
02 kiro_ryoukin = 300 # 1km ごとの料金
03 hatsunori_kyori = 2 # 初乗り距離（初乗り料金のみの距離 :km）
04 jousya_ryoukin = 0
05
06 jousya_kyori = int(input(' 乗車した距離を整数値で入力してください ( 例 : 6km の場合は 6 と入力 ): '))
07
08 if jousya_kyori > hatsunori_kyori:
09   jousya_ryoukin = (jousya_kyori - hatsunori_kyori) * kiro_ryoukin + hatsunori_ryoukin
10 else:
11   jousya_ryoukin = hatsunori_ryoukin
12
13 print(' 乗車料金 ', jousya_ryoukin, ' 円 ')
```

実行結果

```
乗車した距離を整数値で入力してください ( 例 : 6km の場合は 6 と入力 ): 5
乗車料金 1400 円
```

# 上級編　第 36 講　確認問題　解答

正解：①

解説：

if-elif-else 文では，電気の利用量ごとに場合分けして，それぞれの割引率を指定する処理を行っている。ただし，割引率が異なっても，電気利用料金を求める式は，同様に次の通りとなる。

（基本料金（500 円）＋ 電気の利用時間 × 1kWh あたりの価格（100 円））×（1 - 割引率）

よって，正解は①(kihon_ryoukin + ryoukin_kwh * denryoku) * (1 - discount_rate) である。

```
kihon_ryoukin = 500 # 基本料金
ryoukin_kwh = 100 # 1kWh あたりの価格
discount_rate = 0.00

denryoku = int(input('使用した電力量(kWh)を整数値で入力してください(例：200kWhの場合は200と入力)：'))

if denryoku < 100:
  discount_rate = 0.00
elif denryoku < 200:
  discount_rate = 0.10
else:
  discount_rate = 0.15

denki_ryoukin = (kihon_ryoukin +  ryoukin_kwh * denryoku) * (1 - discount_rate)

print('電気料金：', int(denki_ryoukin), '円')
```

実行結果

```
使用した電力量(kWh)を整数値で入力してください(例：200kWhの場合は200と入力)：200
電気料金： 17425 円
```

# 上級編　第 37 講　確認問題　解答

正解：②

解説：

9 行目の if 文の条件式が True であるとき，10 行目では，最高点を表す変数 saikou に Tensuu[i] の値を代入して，最高点の値を上書きする処理を行っている。つまり，この処理を行う条件として，暫定（仮）の最高点よりも，取り出した Tensuu[i] の値のほうが大きい場合を指定すればよいと考えられる。

よって，正解は② `Tensuu[i] > saikou` である。

```
01 Tensuu = [85, 90, 78, 88, 76, 94, 89, 75, 83]
02
03 num = len(Tensuu)
04 goukei = 0
05 saikou = 0
06
07 for i in range(num):
08   goukei = goukei + Tensuu[i]
09   if Tensuu[i] > saikou:
10     saikou = Tensuu[i]
11
12 print(' 合計点 :', goukei, ' 点 ')
13 print(' 最高点 :', saikou, ' 点 ')
```

実行結果

```
合計点： 758 点
最高点： 94 点
```

# 上級編　第38講　確認問題　解答

正解：③

解説：

問題文では,「自分の手の数字と相手の手の数字を使って,ある計算を行う」と結果は表の通りになり，その計算結果は変数 keisan に代入する，と指定されている。そのため，７行目の空欄には，ある計算式が当てはまることになる。

この場合，選択肢①～④の計算式に，各じゃんけんの手の値（グー :0，チョキ :, 1 パー :2）を代入して計算し，どの選択肢が表と同じ計算結果になるか確かめてみるとよい。 よって, 正解は③ (jibun - aite + 3) % 3 である。

```
01 import random
02 
03 Janken = ['グー', 'チョキ', 'パー']
04 print('出す手の数字を入力してください。グー:0, チョキ:1, パー：2')
05 jibun = int(input())
06 aite = random.randint(0,2)
07 keisan = (jibun - aite + 3) % 3
08 
09 if keisan == 2:
10   print('あなたの勝ち')
11 elif keisan == 1:
12   print('あなた負け')
13 else:
14   print('引き分け')
15 
16 print('自分の手：', Janken[jibun], '相手の手：', Janken[aite])
```

実行結果

```
出す手の数字を入力してください。グー：0、チョキ：1、パー：2
0
あなたの勝ち
自分の手： グー　相手の手： チョキ
```

# 上級編　第 39 講　確認問題　解答

正解：④

解説：

8 行目の for 文では，設定した試行回数分，出目 1 と出目 2 の値をランダムで生成する繰り返し処理を行っている。

11 行目からの if 文では，シミュレーション中に，出目 1 と出目 2 の合計が 8 となったとき，変数 deme_kaisuu に 1 を加算している。そのため，100,000 回のシミュレーションが終わったとき（8 行目からの for ブロックを抜けたとき），変数 deme_kaisuu には，出目の合計が 8 になった回数が代入されているといえる。

したがって，14 行目では，「出目の合計が 8 になった回数 ÷ 試行回数 × 100」となる式を当てはめればよいことがわかる。よって，正解は④ `kakuritsu = deme_kaisuu / kaisuu * 100` である。

```
01 import random
02 
03 kaisuu = 100000 # サイコロを振る回数
04 deme_kaisuu = 0 # 出目の回数を格納する
05 goukei = 8 # 確率を求めたい合計値（今回は 8）
06 kakuritsu = 0 # 出目の合計値が出る確率を格納する
07 
08 for i in range(kaisuu):
09   deme1 = random.randint(1, 6)
10   deme2 = random.randint(1, 6)
11   if (deme1 + deme2) == goukei:
12     deme_kaisuu = deme_kaisuu + 1
13 
14 kakuritsu = deme_kaisuu / kaisuu * 100
15 
16 # 結果を表示
17 print('出目の合計', goukei, ':', deme_kaisuu, '回',
   '確率:', kakuritsu, '%')
```

実行結果

```
出目の合計 8 : 13936 回  確率: 13.936000000000002 %
```

# 上級編　第40講　確認問題　解答

正解：③

解説：

左側のグループのデータ数が，奇数個のときと偶数個のときで，処理を分ける必要がある。

13行目の`if left_n % 2 == 0`では，左側のグループのデータ数が偶数個であるかを判定している。よって，この条件式がTrueであるとき，問題文の条件４つ目のとおり，左側のグループデータを真ん中で分割したときの，前後２つの数の平均を求めればよい。 そのため，14行目には，「グループデータの要素数を真ん中で分割したときの，前の数（真ん中の前の数）と，その次の数（真ん中の後ろの数）を足して，２で割る式」が当てはまると考えられる。よって，正解は③ `(Left_data[left_n // 2 - 1] + Left_data[left_n // 2]) / 2`である。

```
01 Data = [1, 4, 6, 8, 9, 10, 13, 15] # 昇順のデータ配列
02 Left_data = [] # 左側のグループのデータを入れる配列
03 n = len(Data)
04 midpoint = n // 2
05
06 # for 文を使用して，左側のグループデータを作成
07 for i in range(0, midpoint):
08   Left_data.append(Data[i])
09
10 left_n = len(Left_data)
11
12 # 左側のデータセットから中央値を取得
13 if left_n % 2 == 0:
14   first_quartile = (Left_data[left_n // 2 - 1] + Left_data[left_n // 2]) / 2
15 else:
16   first_quartile = Left_data[left_n // 2]
17
18 print(int(first_quartile))
```

実行結果

```
5
```

# 上級編　第 41 講　確認問題　解答

正解：①

解説：

13 行目からの if 文では，変数 max_gcd が 1 のとき 2 つの値は素であり，それ以外のとき，2 つの値は素でないことが指定されている。 よって，10 行目に `max_gcd = i` と記述されていることから，空欄は 2 つの値が素であるかを求める式だと分かる。

素であるか確かめるには，2 つの数がともに，2 から，2 つの数のうち小さい数までの整数のいずれかで割りきれるかを判定すればよい。よって，正解は① `number1 % i == 0 and number2 % i == 0` である。

```
01 number1 = int(input('1 つ目の 2 以上の自然数 ( 正の整数 ) を入力してください。'))
02 number2 = int(input('2 つ目の 2 以上の自然数 ( 正の整数 ) を入力してください。'))
03 
04 smallest = min(number1, number2)
05 max_gcd = 1 # 互いに素の場合，最大公約数は 1
06 
07 # 2 つの数の公約数があるか調査
08 for i in range(2,smallest + 1):
09   if number1 % i == 0 and number2 % i == 0:
10     max_gcd = i
11 
12 # 2 つの数が互いに素であるかを判定
13 if max_gcd == 1:
14   print(number1, ' と ', number2, ' は互いに素です。')
15 else:
16   print(number1, ' と ', number2, ' は互いに素ではありません。')
17   print(' 最大公約数は ', max_gcd, ' です。')
```

実行結果

```
1 つ目の自然数 ( 正の整数 ) を入力してください。8
2 つ目の自然数 ( 正の整数 ) を入力してください。36
8 と 36 は互いに素ではありません。
最大公約数は 4 です。
```

# 上級編　第 42 講　確認問題　解答

正解：①

解説：

8 行目からの if 文では，if 条件式が False であるとき（10 行目の else 文の処理にうつるとき)，素数を表す変数 i を配列 Factors の末尾に追加している。ここから，else 文では，変数 i が入力した値の素因数である場合を条件としていることが分かる。

つまり，8 行目の if 文では，変数 i がその整数の素因数でないときの条件式を指定すればよい。よって，正解は ① `number % i != 0` である。

```
01 number = int(input('素因数分解を行う整数を入力してください。'))
02 
03 original_number = number # number の初期値を保存しておく
04 i = 2 # 素数の候補を一時的に格納（2 は最小の素数）
05 Factors = [] # 因数分解した値を格納する配列
06 
07 while i * i <= number:
08   if number % i != 0:
09     i = i + 1
10   else:
11     number = number // i
12     Factors.append(i) # 引数の値を配列 Factors に追加する
13 if number > 1: # 空の配列にならないための条件
14   Factors.append(number)
15 
16 print(original_number, 'の素因数分解：', Factors)
```

実行結果

```
素因数分解を行う整数を入力してください。10
10 の素因数分解：[2, 5]
```

# 上級編　第 43 講　確認問題　解答

正解：④

解説：

2 行目の if 文の条件式が True であるとき，FizzBuzz と表示される。問題文には，FizzBuzz と表示するのは，「3 の倍数かつ 5 の倍数のとき」と指定があるため，if 文の条件式には ④ `i % 3 == 0 and i % 5 == 0` が当てはまることがわかる。

```
01 for i in range(1, 101):
02   if i % 3 == 0 and i % 5 == 0:
03     print('FizzBuzz')
04   elif i % 3 == 0:
05     print('Fizz')
06   elif i % 5 == 0:
07     print('Buzz')
08   else:
09     print(i)
```

実行結果

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
...
```

# 上級編　第 44 講　確認問題　解答

正解：①

解説：

問題文からは，「並び替えるデータの中から最小値を選び，それを配列の左側に入れていくことで，配列内の順番を昇順に並び替える」処理を行うことがわかる。つまり，「仮に指定した配列 Data 内の最小値」と，「配列 Data から順番に取り出した要素の値」の大小を比較して，取り出した値のほうが小さかった場合は，最小値の値を入れ替える処理を行えばよい。

8 行目の if 文の条件式が True であるとき，9 行目にて，変数 min_index に変数 j の値を代入することで，配列 Data の最小値の要素を表す添字を入れ替えている。したがって，8 行目の if 文の条件式には，「変数 j が指定する配列 Data 内の要素のほうが，変数 min_index が指定する配列 Data の要素より小さい場合の式」を指定すればよいことがわかる。

よって，正解は ① `Data[j] < Data[min_index]` である。

```
01 Data = [64, 34, 25, 12, 22, 11, 90]
02 n = len(Data)
03
04 for i in range(n - 1):
05   min_index = i
06   for j in range(i + 1, n):
07     if Data[j] < Data[min_index]:
08       min_index = j
09
10   # 交換処理
11   temp = Data[i]
12   Data[i] = Data[min_index]
13   Data[min_index] = temp
14
15 print('並べ替え後：',Data)
```

実行結果

```
並べ替え後：[11, 12, 22, 25, 34, 64, 90]
```

# 上級編　第 45 講　確認問題　解答

正解：③

解説：

このプログラムでは，⑴入力した人がフォローしている人，⑵入力した人をフォローしている人の２パターンを求めて表示する必要がある。

まず，16 行目の if 文は，⑴を求める（入力した人は誰をフォローしているか）ための判定式である。

そのため，次の if 文である 18 行目では，⑵を求める（入力した人を誰がフォローしているか）ための条件式を指定する必要がある。

よって，２次元配列の１つ目の添字には，表の上から何行目であるかを指定する変数 i が，２つ目の添字には，入力した人の列を指定する [name_index] が当てはまる。

よって，正解は ③ `Data[i][name_index] == 1` である。

```
Data = [
  [0, 1, 1, 0], # A
  [1, 0, 0, 1], # B
  [0, 0, 0, 0], # C
  [1, 1, 0, 0]  # D
]

Human = ['A', 'B', 'C', 'D']
ToFollow = [] # フォローしている人を格納
FromFollow = [] # フォローされている人（フォロワー）を格納

name = input('AからDの中で，フォロー関係を知りたい人を入力してください。（例：B）： ')
name_index = Human.index(name) # 入力された人に対応する添字を取得

for i in range(len(Data)):
  if Data[name_index][i] == 1:
    ToFollow.append(Human[i])
  if Data[i][name_index] == 1:
    FromFollow.append(Human[i])

# 結果表示
print(name, 'がフォローしている人は')
print(ToFollow)
print(name, 'をフォローしている人は')
print(FromFollow)
```

実行結果

```
AからDの中で、フォロー関係を知りたい人を入力してください。
（例：B）： B
B がフォローしている人は
['A', 'D']
B をフォローしている人は
['A', 'D']
```

# 上級編　第 46 講　確認問題　解答

正解：①

解説：

if 文の条件式に True か False のどちらかを持つ bool 型を指定すると，その値が True のときに if 文内の処理に入り，False のときは else 文内の処理に入る。つまり，17 行目の `if blue_flag:` では，bool 型の `blue_flag` が True のときは if 文内の処理に入り，False のときは else 文内の処理に入る。

次に if 文内，空欄のある 18 行目を見ていく。変数 wait には，その青信号の時間内に通過しきれなかった車の台数を代入すればよい。

よって正解は① `wait + arrive - passtime` である。

なお，青信号の時間内にすべての車が通過できた場合は，`wait + arrive - passtime` が 0 以下となるため，max 関数によって，もう一つの引数である 0 が返され，変数 wait に代入される。

```python
import random

totaltime = 18   # 観測時間：単位 10 秒（180 秒なら 18）
passtime = 10    # 通過可能台数
signal_b = 6     # 青信号の時間：単位 10 秒
signal_r = 3     # 赤信号の時間：単位 10 秒
blue_flag = True # 青信号の場合「True」
arrive = 0       # 車の到着台数
wait = 0         # 渋滞台数

print(' 秒　到着台数　渋滞台数 ')

for t in range(1, totaltime + 1):
  arrive = random.randint(5, 10) # 10 秒間に 5 ～ 10 台ランダムに車が到着する
  mod = (t - 1) % (signal_b + signal_r) + 1
  blue_flag = (mod <= signal_b)
  if blue_flag: # 青信号ならば
    wait = max(wait + arrive - passtime, 0)
  else: # 赤信号ならば
    wait = wait + arrive # 通過できない車はすべて渋滞に加算される
  print(t * 10, '¥t', arrive, '¥t', wait, '¥t')
```

実行結果

| 秒 | 到着台数 | 渋滞台数 |
|---|---|---|
| 10 | 5 | 0 |
| 20 | 9 | 0 |
| 30 | 9 | 0 |
| 40 | 8 | 0 |
| 50 | 9 | 0 |
| 60 | 10 | 0 |
| 70 | 6 | 6 |
| 80 | 9 | 15 |
| 90 | 8 | 23 |
| 100 | 6 | 19 |
| 110 | 10 | 19 |
| 120 | 8 | 17 |
| 130 | 6 | 13 |
| 140 | 8 | 11 |
| 150 | 6 | 7 |
| 160 | 10 | 17 |
| 170 | 5 | 22 |
| 180 | 7 | 29 |

# 上級編　第 47 講　確認問題　解答

正解：②

解説：

問題文のチェックディジットの求め方の手順⑴⑵から，左から数えたとき，奇数番目になる桁のときと，偶数番目になる桁のときで処理を分ける必要があることが分かる。

11 行目の if 文では，`if i % 2 == 0` とあるので，これは Jan_code の添字 i が偶数（0 番目，2 番目…），つまり，桁数が奇数桁（1 桁目, 3 桁目…）の条件を指定している。この条件が True であるとき，奇数の桁数の和を表す変数 odd_sum に奇数桁の値を加算している。

13 行目 else 文では，桁数が偶数桁の場合となるので，偶数桁の値を加算して，偶数の桁数の和を表す変数 even_sum に代入している。ここまでの処理で，奇数桁の合計値 odd_sum と偶数桁の合計値 even_sum が求められた。

続いて，問題文のチェックディジットの求め方の手順⑶では，「⑴の結果と⑵の結果の和を求める」とあるため，`odd_sum + even_sum` の式となる。ただし, 注意したいのは，手順⑵で偶数桁の合計値を 3 倍する，と指定があるためこの式を `odd_sum + even_sum *3` とする必要がある。よって，正解は ② `odd_sum + even_sum * 3` である。

```python
Jan_code = input('13 桁の JAN コードを入力してください : ')

if len(Jan_code) != 13:
  print(' 入力されたコードは 13 桁ではありません。')
  exit()

odd_sum = 0 # 奇数桁の合計値を格納
even_sum = 0 # 偶数桁の合計値を格納

for i in range(12):
  if i % 2 == 0:
    odd_sum = odd_sum + int(Jan_code[i])
  else:
    even_sum = even_sum + int(Jan_code[i])

total = odd_sum + even_sum * 3

ones_place = total % 10

if ones_place == 0:
  check_digit = 0
else:
  check_digit = 10 - ones_place

# チェックディジットの検証
if check_digit == int(Jan_code[-1]):
  print('JAN コードは正しいです。チェックディジット :', check_digit)
else:
  print('JAN コードは誤っています。正しいチェックディジットは ', check_digit, ' です。')
```

実行結果

```
13 桁の JAN コードを入力してください : 1234567890123
JAN コードは誤っています。正しいチェックディジットは 8 です。
```

# 上級編　第 48 講　確認問題　解答

正解：③

解説：

19 行目の if 文の条件式が空欄となっている。この if 文では，プレイヤーがお化けのマスに止まったときの処理を記述している。このことから，空欄には，プレイヤーのいるマスとお化けのいるマスが同じかどうか判定する条件式が入ることが分かる。

よって，正解は③ `player_position == ghost_position` である。

```
01 import random
02 
03 # ゲームの基本設定
04 goalposition = 20
05 Ghostpositions = [15, 16, 17]  # お化けがいる可能性のある位置
06 player_position = 0  # プレイヤーの初期位置
07 roll_count = 0  # サイコロを振った回数
08 
09 # ゲーム開始
10 while player_position < goalposition:
11   input('サイコロを振ってください(Enterを押す)！')
12   roll = random.randint(1, 6)
13   player_position = player_position + roll
14   roll_count = roll_count + 1
15   print('サイコロの出目：', roll, ' ', player_position, 'マス目に移動しました。')
16 
17   # お化けのチェック
18   ghost_position = random.choice(Ghostpositions)
19   if player_position == ghost_position:
20     print('お化けに遭遇！5マス戻ります。')
21     player_position = player_position - 5
22 
23 # ゴールに到達
24 print('ゴールに到達しました！おめでとうございます！')
25 print('合計でサイコロを', roll_count, '回振りました。')
```

実行結果

```
サイコロを振ってください(Enter を押す)！
サイコロの出目： 6    6 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目： 3    9 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目： 2    11 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目： 5    16 マス目に移動しました。
お化けに遭遇！ 5 マス戻ります。
サイコロを振ってください(Enter を押す)！
サイコロの出目： 5    16 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目： 3    19 マス目に移動しました。
サイコロを振ってください(Enter を押す)！
サイコロの出目： 5    24 マス目に移動しました。
ゴールに到達しました！おめでとうございます！
合計でサイコロを 7 回振りました。
```

# 商標について

・Python は，Python Software Foundation の商標または登録商標です。

・その他，本講座で扱う社名，商品名，サービス名等は，一般に各社の商標または登録商標です。

・なお，本講座では，(R)，TM マークは明記しておりません。

Ⓒ RECRUIT
本サービスに関する知的財産権その他一切の権利は著作権者に帰属します。
また本サービスに掲載の全部または一部につき無断複製・転載を禁止します。